IDENTIFICACION DE LOS DATOS DEL PROYECTO

TEMA	Procesado de señales acústicas
TITULO	Codificador de voz MIDI
AUTOR	Oliver Vivar Meliveo
TITULACIÓN	Ing. Tec. Telecomunicaciones Esp. Sonido e Imagen
TUTOR	Danilo Simón Zorita
DEPARTAMENTO	DIAC
DIRECTOR	Danilo Simón Zorita

TRIBUNAL	PRESIDENTE	Antonio Redondo Hidalgo
	VOCAL	Danilo Simón Zorita
	VOCAL SECRETARIO	Lino García Morales
	FECHA DE LECTURA	Febrero 2013

AGRADECIMIENTOS

A mi madre, sin ninguna duda.

A vosotros y al K, por todos esos dias de mísica que sembraron en mi la idea de este proyecto.

A esta escuela que no solo me ha dado conocimientos técnicos, sino una forma de ver la vida. Y a mi tutor, por darme la libertad de elegir mi propio camino.

Creo que quien ha disfrutado con los sublimes placeres de la música, deberá ser eternamente adicto a este arte supremo y jamás renegar de él.

Richard Wagner

Cuando creas en ti, ni el cielo será tu límite

Miles Davis

En la vida hay tres tipos de hombres, los que saben contar y los que no

Homer J. Simpson

RESUMEN

El presente proyecto tiene el objetivo de facilitar la composición de canciones mediante la creación de las distintas pistas MIDI que la forman. Se implementan dos controladores.

El primero, con objeto de transcribir la parte melódica, convierte la voz cantada o tarareada a eventos MIDI. Para ello, y tras el estudio de las distintas técnicas del cálculo del tono (pitch), se implementará una técnica con ciertas variaciones basada en la autocorrelación. También se profundiza en el segmentado de eventos, en particular, una técnica basada en el análisis de la derivada de la envolvente.

El segundo, dedicado a la base rítmica de la canción, permite la creación de la percusión mediante el golpe rítmico de objetos que disponga el usuario, que serán asignados a los distintos elementos de percusión elegidos. Los resultados de la grabación de estos impactos serán señales de corta duración, no lineales y no armónicas, dificultando su discriminación. La herramienta elegida para la clasificación de los distintos patrones serán las redes neuronales artificiales (*RNA*).

Se realizara un estudio de la metodología de diseño de redes neuronales especifico para este tipo de señales, evaluando la importancia de las variables de diseño como son el número de capas ocultas y neuronas en cada una de ellas, algoritmo de entrenamiento y funciones de activación. El estudio concluirá con la implementación de dos redes de diferente naturaleza.

Una red de Elman, cuyas propiedades de memoria permiten la clasificación de patrones temporales, procesará las cualidades temporales analizando el ataque de su forma de onda.

Una red de propagación hacia adelante *feed-forward,* que necesitará de robustas características espectrales y temporales para su clasificación. Se proponen 26 descriptores como los derivados de los momentos del espectro: centroide, curtosis y simetría, los coeficientes cepstrales de la escala de Mel (*MFCCs*), y algunos temporales como son la tasa de cruces por cero y el centroide de la envolvente temporal. Las capacidades de discriminación inter e intra clase de estas características serán evaluadas mediante un algoritmo de selección, habiéndose elegido RELIEF, un método basado en el algoritmo de *los k vecinos mas próximos* (KNN).

Ambos controladores tendrán función de trabajar en tiempo real y *offline*, permitiendo tanto la composición de canciones, como su utilización como un instrumento más junto con mas músicos.

ABSTRACT

The aim of this project is to make song composition easier by creating each MIDI track that builds it. Two controllers are implemented.

In order to transcribe the melody, the first controler converts singing voice or humming into MIDI files. To do this a technique based on autocorrelation is implemented after having studied different pitch detection methods. Event segmentation has also been dealt with, to be more precise a technique based on the analysis of the signal's envelope and it's derivative have been used.

The second one, can be used to make the song's rhythm . It allows the user, to create percussive patterns by hitting different objects of his environment. These recordings results in short duration, non-linear and non-harmonic signals. Which makes the classification process more complicated in the traditional way. The tools to used are the artificial neural networks (*ANN*).

We will study the neural network design to deal with this kind of signals. The goal is to get a design methodology, paying attention to the variables involved, as the number of hidden layers and neurons in each, transfer functions and training algorithm. The study will end implementing two neural networks with different nature.

Elman network, which has memory properties, is capable to recognize sequences of data and analyse the impact's waveform, precisely, the attack portion.

A *feed-forward* network, needs strong spectral and temporal features extracted from the hit. Some descriptors are proposed as the derivates from the spectrum moment as centroid,

kurtosis and skewness, the Mel-frequency cepstral coefficients, and some temporal features as the zero crossing rate (*zcr*) and the temporal envelope's centroid. Intra and inter class discrimination abilities of those descriptors will be weighted using the selection algorithm RELIEF, a *Knn* (K-nearest neighbor) based algorithm.

Both MIDI controllers can be used to compose, or play with other musicians as it works on real-time and offline.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

PARTE I

Planteamiento del proyecto y revisión de conocimientos

Capítulo 1

Introducción

1.1 Introducción general, motivación y objetivo del proyecto.

La música es una de las artes con mayor difusión que existe, siendo para algunos una fuente inagotable de felicidad.

En todo arte existe el autor o autores de la obra, y todo aquel espectador u oyente que quiera disfrutar de la misma. Este último se convierte en un sujeto pasivo, pero secretamente quiere ser partícipe de la creación, quiere crear su propio arte y disfrutar de esa sensación.

Pero, ¿es tan difícil la creación musical?. En el camino de llegar a tener la posibilidad de crear música uno puede encontrarse muchos obstáculos que pueden esquivarse eligiendo un instrumento, formándose en escuelas de música o conservatorios etc. Pero esto es algo que requiere tiempo y sacrificio.

Afortunadamente, con la creación del MIDI (*Musical Instrument Digital Interface*), las fronteras entre los instrumentos se difuminaron enormemente, pudiendo tocar un teclado y asignarle el instrumento que se desee. Pero aun así, hay que saber tocar el teclado para disfrutar de estas facilidades.

Posteriormente se crearon dispositivos hardware y software de transcripción a MIDI con diferentes fuentes, ya fuese guitarra, vientos, voz… con posibilidades de trabajar en tiempo real. La investigación en este área prosigue, donde el tiempo de procesado obliga a realizar rápidos algoritmos que fallan en precisión.

La finalidad de este proyecto, es el de implementar un sistema que permita al usuario la creación de una obra musical en su hogar independientemente del conocimiento musical que éste tenga.

Este objetivo requiere la distinción de dos familias de instrumentos en la composición musical, los dedicados a la parte melódica (guitarra, bajo, saxofón etc.) y los encargados de soportar la parte rítmica (batería, bongos etc.), en esta última familia el concepto de notas, no está tan claro.

Por esto se van a implementar dos sistemas distintos:

- La traducción de la parte melódica se realizará extrayendo las notas producidas por la voz y transcribiéndolas a MIDI. Por lo que el usuario solo deberá tararear las melodías y asignarlas al los instrumentos que más le gusten.

- Para la traducción rítmica, el usuario deberá improvisar una batería en su propio hogar, un par de lapiceros usados como baquetas y una mesa sobre la que interpretar los distintos golpes sobre superficies que él elija constituirán una improvisada batería. Mediante redes neuronales artificiales, y tras una fase de entrenamiento, el sistema distinguirá cada uno de estos sonidos asignándole un elemento de la familia de instrumentos percusivos.

De esta forma se busca disminuir la distancia entre la imaginación y la interpretación musical.

1.2 Estructura y contenido del PFC

En este apartado se describe brevemente la organización de contenidos en los que se divide el proyecto.

I. Planteamiento del proyecto y revisión de contenidos: capítulos 1-3.

En esta parte se persigue sentar las bases sobre las que construir el sistema. Se repasarán los sistemas de transcripción a lo largo de la historia y en la actualidad en el capítulo 1. Para ahondar más específicamente en los dos tipos de transcripciones que abarca este proyecto: la transcripción melódica de voz (capitulo 2), y la transcripción rítmica de sonidos sin tonalidad definida (capitulo 3). En este último se introducen los fundamentos de las redes neuronales.

II. Desarrollo del proyecto: capítulos 4-6.

Representa el trabajo desarrollado para cumplir los objetivos previamente mencionados. Se abordará la transcripción de voz (capítulo 4) y su problemática, así como la transcripción rítmica (capítulo 5), para cual se realizara un estudio de una metodología de diseño de redes neuronales artificiales, para diseñar dos redes con diferentes propiedades.

Finalmente se procederá a explicar el entorno grafico (capítulo 6) que permite la navegación del usuario a través de las posibilidades que ofrece el sistema implementado.

III. Resultados, conclusiones y líneas futuras de desarrollo: capito 7.

En esta sección se prueban los diferentes sistemas implementados, evaluando las capacidades de los mismos. Así como la utilidad y eficiencia de las herramientas y procedimientos utilizados.

Finalmente se propondrán mejoras, dándose una visión personal del futuro de la transcripción musical.

1.3 Herramienta de trabajo

Como principal herramienta para la implementación del sistema se utiliza el software de programación de alto nivel Matlab de Mathworks, que hace uso del lenguaje propio *M*.

Esta plataforma no es la más adecuada para aplicaciones en tiempo real, pero permite dar una buena idea en las primeras fases de desarrollo, para luego implementarse en un lenguaje de bajo nivel como pueda ser C++.

1.4 Estado del arte

1.4.1 Historia

La historia de la transcripción automática comienza a principios de la década de 1970, con la creación de un sistema de transcripción de duetos por parte de Moorer [1]. El sistema tenía ciertas limitaciones, ya que las dos líneas melódicas debían estar en frecuencias diferentes y los intervalos entre notas debían tener una distancia mínima.

A finales de los años 80 la universidad de Osaka, comenzó un ambicioso e interesante proyecto, el cual tenía como objetivo el crear un robot capaz de extraer sentimientos de la música, imitando a un oyente humano [2]. Para lograr este objetivo, se encontraron con la necesidad de realizar una transcripción musical, creando dos sistemas. Uno de ellos para canciones monofónicas tradicionales japonesas, y otro polifónico (hasta 5 voces) para interpretaciones de piano, guitarra o samisén (instrumento de cuerda tradicional japonés).

Tras Moorer, se crearon varios sistemas para la transcripción automática de melodías polifónicas, en 1993 Hawley consiguió la transcripción exitosa de melodías interpretadas por piano, mediante el estudio diferencial de FFTs consecutivas [3]. El piano, al no poder modificar el tono de una nota, es más fácil de transcribir, teniendo el problema de la multitonalidad de los acordes.

Un avance significativo en la transcripción musical polifónica se dio en la Universidad de Tokio, donde se introdujeron nuevas técnicas [4]. Siendo los primeros en utilizar criterios psicoacústicos en la discriminación o fusión de componentes espectrales próximos. También introdujeron en el procesado, información sobre los instrumentos, proponiendo un algoritmo de análisis automático. En 1995 se mejoró el sistema mediante la arquitectura de pizarra (*Blackboard Arquitecture*).

En cuanto a la transcripción rítmica, la mayoría de los estudios se han basado en la transcripción de instrumentos de percusión con tono, como puedan ser una marimba o un metalófono, aunque debido a la importancia de la batería en la música popular actual, también se está profundizando en la transcripción de instrumentos sin tono definido.

Schloss [5] en el año 85, ya creó un sistema de transcripción rítmica, que trabajaba sobre un tramo improvisado interpretado por una conga. El sistema segmentaba las notas en base a la derivada de la envolvente temporal. Realizando la clasificación del sonido mediante la comparación de la energía en las diferentes sub bandas del tramo de señal, con la distribución obtenida en la fase de entrenamiento.

Más tarde Goto y Muraoka [6] con el objetivo de obtener un seguimiento del tempo en composiciones polifónicas, propusieron una segmentación basada en la localización de incrementos de energía en el dominio de la frecuencia.

El procedimiento de estos sistemas suele constar de dos fases. Primero la detección de eventos y etiquetado de los mismos usualmente siguiendo las ideas de Klapuri [7]. Y posteriormente obteniendo las características que se consideren sean decisivas en la clasificación. En [8] se puede apreciar una comparación de las diferentes características de instrumentos de percusión, los cuales fueron analizados individualmente.
Finalmente, se realiza la clasificación basándose en modelos Gausianos o los modelos ocultos de Markov entre otras técnicas.

El estado del arte de los transcriptores musicales está, en la actualidad, claramente por debajo de las habilidades y precisión de un oído humano entrenado en el arte de la música.

1.4.2 Tecnologías actuales.

La industria de software musical se ha percatado del mercado de la transcripción musical. La gran mayoría de los sistemas comerciales proponen la conversión al lenguaje MIDI (como se ha elegido en este proyecto).

Algunos eligen soluciones basadas en VST, de las que hacen uso secuenciadores y estaciones de audio digitales como podría ser Cubase, Ableton Live, Pro-tools etc. Y otras proponen soluciones independientes.

Se pasa a describir algunos de los sistemas de transcripción automática que existen actualmente:

Transcripción Melódica

TwelveKeys Music Transcription Software

Este software de transcripción melódica polifónica, es muy útil si el usuario parte de cierto conocimiento musical, para poder componer acordes y melodías más fácilmente. Este permite variar la desafinación de todo el archivo de entrada. Es decir, si se importa un archivo grabado con una guitarra que no estuviera afinada en el LA 440 Hz, o más comúnmente la voz, se podría variar la detección para disminuir el número de errores.

Seventh String Software.

Útil herramienta de transcripción offline, la cual ofrece una interesante interfaz grafica que permite observar la distribución espectral de cualquier segmento en función de un piano. Tanto este software como *TwelveKeys* necesitan de un pre procesado, en el caso de grabaciones polifónicas, teniéndose que eliminar la voz y elementos percusivos que puedan entorpecer la detección.

Widisoft

Potente herramienta que permite la transcripción tanto de ficheros grabados, como de entradas en tiempo real.

Los parámetros a introducir por el usuario no son excesivos, y a la hora de transcribir monofonía, los resultados son excelentes, casi sin error, pero la tasa de error empieza a elevarse cuando tratamos de convertir sonidos polifónicos.

Existen varios programas similares a Widisoft como son digitalEar, Intelliscore o Amazing MIDI, los cuales realizan las mismas tareas con mayor o menor acierto. Pero sin duda, el programa que presenta grandes cambios con los anteriores es Melodyne de Celemony

Celemony Melodyne

Esta poderosa herramienta presenta cada melodía como una secuencia segmentada de notas, con la posibilidad de tratar cada nota de forma independiente. Cada nota de un acorde (o cualquier melodía polifónica). En este caso, el programa no realiza una conversión a MIDI (aunque también ofrece esta posibilidad), sino que simplemente divide las notas de audio e

identificando que nota es. Las posibilidades son impresionantes, pudiendo usarse como instrumento virtual. Este programa es comúnmente usado para el procesado de voz en estudio, para afinar interpretaciones etc.

El objetivo de este programa no es el de que una guitarra suene como una trompeta, está más orientado a la edición, para lo cual la transcripción también es necesaria.

Sistemas QbH (Query by Humming)

Estos software que usualmente tienen formato de aplicaciones para dispositivos móviles, permiten al usuario el tararear (*Humm*) al micrófono un fragmento de una canción, y que éste encuentre, mediante una comparación de características con una base de datos, el nombre de la canción y el artista que la interpreta.

El objetivo es acabar con esa molesta sensación de tener una canción en la cabeza y no poder escuchar la original, o no saber a qué artista pertenece.

Ejemplos de este tipo de sistemas son Midomi y SoundHound, el primero ofrece un servicio online sin necesidad de realizar descarga. SoundHound existe únicamente para dispositivos móviles.

Transcripción Rítmica

Drumagog

Este software tiene como objetivo que el usuario pueda tocar la batería sin la necesidad de tener una batería física. Es decir, asocia cualquier señal percusiva que esté grabando el micrófono a un único elemento de la batería. El problema que presenta es que son necesarios tantos micrófonos como elementos de la batería queramos tocar, por lo que si se quisiera interpretar un ritmo básico de batería que involucrase bombo, caja, charles y crash (platillo) serian necesarios 4 micrófonos.

Al no realizar prácticamente ningún procesado, la velocidad de respuesta se reduce enormemente, operando magníficamente en tiempo real.

1.5 MIDI (Musical Instrument Digital Interface)

Los datos de audio que van a ser procesados, tanto rítmica como melódicamente van a ser traducidos a MIDI. Se pasa a introducir brevemente este sistema.

MIDI es un protocolo de comunicación serie que permite a una amplia variedad de dispositivos como ordenadores, secuenciadores y sintetizadores la posibilidad de compartir información mediante un mismo lenguaje para generar los sonidos.

A grandes rasgos, es un lenguaje musical que trata de convertir a datos, las posibles interpretaciones que un músico puede realizar. Esto no es algo nuevo, no es más que la versión informática de una partitura.
Pero en este caso, en vez de existir diferentes elementos en los pentagramas hablaremos de mensajes.

Mensajes

Existe una gran variedad de mensajes, algunos de ellos se muestran en el Anexo II Tabla 8 y Tabla 9. Aunque no todos afectan a la interpretación, en este proyecto solo se van a utilizar los siguientes.

Note On/Note Off

Mediante estos dos mensajes se consigue componer el elemento fundamental de cualquier composición: la nota.
Se enviara el mensaje de *NOTE ON* al iniciarse la nota y *NOTE OFF* al finalizar. Ambos mensajes comparten la misma información a enviar:

- Nota: indica la nota que se está iniciado/finalizando. Se tiene en cuenta diez octavas, asignándole a cada nota un numero de 0-127.
- Velocidad: es la fuerza o volumen que va a tener dicha nota, ya sea al iniciarse o al apagarse (también entre 0-127).
- Canal MIDI: para permitir la interpretación de varios instrumentos al mismo tiempo, la especificación ofrece 16 canales. De esta forma, si configuramos un sintetizador a un canal especifico, este solo reproducirá los mensajes que tengan el código asociado a dicho canal.

Haciendo un símil con un piano, cuando el músico presione un tecla, se enviara Note On, con la nota pulsada y la velocidad con la que se haya oprimido. En el momento en el que levante el dedo de la tecla, se enviara Note Off con la nota que se está finalizando y la velocidad con la que ha levantado el dedo.

Pitch Bend

Permite una modulación en frecuencia del canal especificado. Este mensaje representa la posibilidad de que el músico decida ligar dos notas, o simplemente variar el tono de la interpretación. Por ejemplo, el piano seria un instrumento que carece de la posibilidad de ligar notas, si se fingiera por un momento que se es ese pianista que pasa la mano de un lado a otro del piano rápidamente pulsando todas las teclas, se podrá oír como existen pequeños cambios en frecuencia por cada nota que pasa. Pero en cambio, un violinista puede mover un dedo a lo largo de la cuerda produciendo una variación suave sin estos pequeños cambios.

El mensaje de *Pitch Bend* permite realizar esta interpretación enviando un mensaje con las siguientes características:

- Variación: se asigna a la variación en frecuencia un valor, entre -8192 y 8192 siendo el 0 el tono original, valores positivos producen una variación hacia sonidos agudos y negativos viceversa.
- Canal MIDI: de igual objetivo que en Note On/Off.

Polyphonic Key Pressure (*Aftertouch* o pos-pulsación)

Al igual que Pitch Bend permite la modulación en frecuencia, el mensaje *Aftertouch* (Post-pulsación), representa la modulación en amplitud. Esta interpretación es muy típica de los instrumentos de viento. Por ejemplo, un saxofonista en medio de un solo en el cual, manteniendo una misma nota parte de un volumen muy bajo y asciende para darle dinámica.

El mensaje está compuesto por:

- Nota: la nota sobre la cual se modifica la dinámica.
- Variación de Presión: representa el valor de la post-pulsación, teniendo una precisión entre 0-127.

Capítulo 2

Transcripción melódica

En esta parte del proyecto se busca conseguir una transcripción de la voz humana cantada a eventos MIDI, tanto en tiempo real (*on-line*), como en pos procesado (*off-line*).

Se abarcará el problema de la transcripción melódica de forma general, así como los problemas específicos asociados a la voz cantada. Se dará una visión de conjunto partiendo de una breve descripción de la voz y del aparato fonador, para luego evaluar los métodos de detección de tono así como los de segmentado de eventos.

2.1 La voz

El proceso de producción humana comienza en los pulmones. Cuando el aire contenido en éstos atraviesa los bronquios y la tráquea para llegar a la laringe.
En la laringe existen dos pares de cuerdas vocales, las dos superiores que no participan en la articulación de la voz, y las dos inferiores, responsables de la excitación sonora.

Lo que comúnmente se llama cuerda vocal no es tal cosa, realmente son dos membranas que se cierran o abren. Cuando estas se contraen separándose, el aire proveniente de los pulmones pasa libremente a través de ellas, produciendo sonidos sordos (fricativas etc.). Pero si en cambio las cuerdas ofrecen resistencia al aire que lo atraviesa, estas vibran debido a la obstrucción produciendo sonidos sonoros. Esta vibración es la responsable de la voz.

Entendiendo entonces el mecanismo de excitación productor de la voz como la vibración de una membrana, la tensión, colocación etc. de la misma definirá un modo propio de radiación sonora, a esto se le llamara **tono** de la voz. A esta frecuencia fundamental o tono, se le suman los armónicos de dicha frecuencia, ya que se recuerda que es una membrana vibrando.

Una manera común de diferenciar cuando se está produciendo un sonido sordo o sonoro es medir la tasa de cruces por cero. Esta técnica se basa en la naturaleza ruidosa y aleatoria de los sonidos sordos frente a la periodicidad de los sonoros. Consiste en medir el número de veces que la señal cruza el eje de amplitud cero. En el caso de los sonidos sordos, el número de cruces por cero será mayor.

Esta técnica será utilizada en el algoritmo, no para diferenciar entre sonidos sordos y sonoros sino para distinguir los tramos de voz cantada y ruido ambiente.

El sonido producido por las cuerdas vocales es amplificado por los distintos resonadores nasal, bucal y faríngeo, donde se produce una coloración del sonido (filtrado), conformando los llamados formantes.

Finalmente la forma en que el paladar, los dientes, labios y glotis se posicionen dará lugar a las distintas articulaciones que conforman el habla humana.

2.2 Transcripción: descripción del problema

Generalmente el problema de la transcripción melódica de voz cantada o tarareada se formula de la siguiente manera.

Dada una forma de onda correspondiente a una interpretación cantada, sin otros instrumentos. Se desea obtener una secuencia de notas y silencios que melódica y rítmicamente se parezcan a la voz original lo máximo posible.

Cada nota tiene como parámetros de interés: el tono, un comienzo (*onset*) y un final (*offset*). El tono o frecuencia fundamental (F0) en Hz corresponde al tono de la voz. La relación entre la frecuencia fundamental de dos notas se llama intervalo; un intervalo de $1:2$ es lo que se conoce como octava, siendo el menor intervalo el correspondiente a un semitono $1:2^{1/12}$.

Una de las grandes dificultades de la transcripción de la voz cantada o tarareada es la variación de su tono a lo largo de una misma nota. Obtener la media de todos los valores de tono no parece ser un buen método ya que la curva de F0 muestra grandes desviaciones frente a las notas MIDI. Además de que las interpretaciones cantadas suelen distar mucho de ser precisas. Esta es una de las diferencias fundamentales entre la voz y los demás instrumentos, los cuales presentan mayor estabilidad en el sostenimiento de una nota.

Estas desviaciones de tono con respecto a la nota temperada son, en algunos casos, realizados de forma intencionada aumentando los detalles de la interpretación.

Casos de esta índole son el tremolo o vibrato, en donde se varia periódicamente el tono de la voz (entre 4 y 7Hz). Los resultados experimentales con oyentes muestran que la media de la frecuencia fundamental durante el vibrato es cercana a la interpretación auditiva de la nota [9]. El *glissando* o *legatto* donde el cantante realiza una variación (normalmente ascendente) del tono comenzando algunos semitonos por debajo para luego afinarse a la nota deseada.

Pero el caso más común es que la interpretación sea pobre, estando la nota desafinada sin intención. Los estudios realizados muestran que la voz de un cantante profesional y con acompañamiento suele desviarse ± 0.07 semitonos lo cual es una gran afinación. Pero esta situación es diferente cuando el artista lo interpreta a cappella. En general, los cantantes son incapaces de cantar afinados sin referencia.

Un problema muy común de las interpretaciones sin acompañamiento es que la afinación va variándose gradualmente, y las notas que ocupan el final tendrán una referencia totalmente diferente de las iníciales.

Esto conlleva a plantearse tres distintas suposiciones con respecto a las señales cantadas que van a tratarse.

- El interprete realiza una interpretación perfectamente afinada.
- El interprete tiene una afinación propia que mantiene a lo largo del tramo.
- El interprete varia la entonación conforme va cantando.

Según cada una de estas suposiciones, se explican brevemente tres sistemas que abordan esta problemática.

- **Afinación perfecta**: El sistema que se refleja en [10] tiene como objetivo el transcribir lo más precisamente el pitch en notación musical, en vez de tener en cuenta los deseos del cantante. La frecuencia fundamental es calculada como la media aritmética de la estimación obtenida en la parte central del segmento. Los F0 calculados que difieran un 10% de la media son eliminados, y la media es re calculada. Llegando a transcribirse ésta a la notación MIDI.

- **Afinación constante**: Haus y Pollastri mantienen la postura de que el tramo cantado mantendrá una referencia de afinación constante [11] aunque no sea una nota temperada. Habiendo obtenido los valores de F0 a lo largo de la nota, se realiza un filtro de mediana de 3 puntos para eliminar elementos erróneos. En el caso en que se detectase un *legato,* el tramo bajo estudio se divide en dos y tratados independientemente. El F0 final es calculado como media de los valores obtenidos y r epresentado en MIDI.

  Para el cálculo de la desviación constante, se calcula un histograma de la desviación de cada nota con respecto a la afinación absoluta. El valor de desafinación que más se dé en el histograma indica la desviación más común frente a la referencia absoluta (LA 440 Hz).

- **Afinación variable:** McNab [12] propone un sistema que se adapta continuamente a la afinación de la interpretación. Para la primera nota, se calcula el histograma de F0, y el valor más repetido se transcribe a MIDI. Para la siguiente nota, el histograma es modificado según la desafinación obtenida en la anterior nota. De esta forma, la afinación se variando en cada intervalo.

  Este sistema tiende a afinar en exceso. Siendo su error máximo de medio semitono en un intervalo.

2.2.1 Pre procesado de la señal

El pre procesado tiene como objetivo mejorar la señal de interés para facilitar la extracción de características. Si la voz se encuentra inmersa en un medio ruidoso, esto afectará negativamente. El problema de la reducción de ruido en el entorno de voz ha sido estudiado con detalle en la rama del reconocimiento del hablante, pudiéndose extrapolar estos conocimientos a la voz cantada. Un amplio estudio es realizado en [13], el cual tiene en cuenta diferentes fuentes de ruido, proponiendo sistemas de predicción estadísticos como los modelos ocultos de Markov.

También es común en esta fase, la ecualización del espectro con objetivo de reducir el efecto pernicioso de los formantes. En algunas ocasiones se obtiene el filtro de formantes propio de la voz mediante un predictor lineal, y se aplica un filtrado inverso a la señal de entrada [14] .

2.2.2 Detección de Tono (*Pitch*)

La detección de la frecuencia fundamental o tono, constituye una de las características más importantes a extraer en la transcripción de el canto, ya que de este depende la nota extraída. La obtención de F0 ha sido estudiada ampliamente en el campo del procesado de voz hablada. La mayoría de los algoritmos que se utilizan en la voz cantada, vienen de ser diseñados para el habla.

Se pasa a introducir brevemente algunos de los algoritmos más útiles en la obtención de F0 para la transcripción de voz.

Detección en el dominio del tiempo

Dado que la frecuencia de una señal es la inversa del periodo, y siendo el periodo el tiempo en el cual una señal periódica se repite (Figura 1), una forma de obtener el periodo seria medir la distancia entre cruces por cero, o entre máximos de una señal. Este método tiene sentido en el caso de señales sinusoidales puras, pero en el caso de señales complejas, la estimación del tono se hace prácticamente imposible.

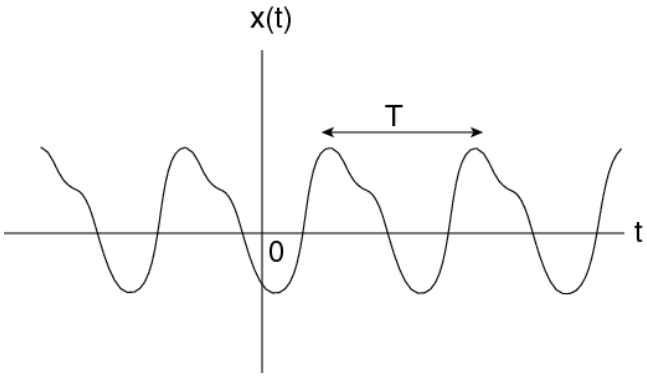

Figura 1: Señal donde se detalla el valor del periodo fundamental T.

El método más usado en el dominio del tiempo es el basado en la autocorrelación.

Autocorrelación

La autocorrelación mide el parecido de una señal con una réplica exacta de ésta, en función del desplazamiento que la réplica va realizando por cada iteración. La idea es medir la correlación entre las dos señales. Encontrando el momento en el cual son más parecidas, sabremos que el desplazamiento ha sido el correspondiente a un periodo.

Dada una señal $s(k)$ muestreada en el dominio del tiempo, una ventana de tamaño W y siendo τ el desplazamiento, la autocorrelación puede expresarse como

$$r_t(\tau) = \sum_{k=t}^{t+W-1} s(k)s(k+\tau) \qquad (1)$$

Para valores de desplazamiento k iguales al periodo y múltiplos de la señal, la función autocorrelación tiene un máximo local, luego buscando la distancia entre máximos, obtendremos el periodo fundamental (ver Figura 2).

Este método es fácil de implementar y de usar, pero un gran inconveniente es la gran sensibilidad de éste a los formantes, teniendo a menudo los llamados errores de octava. Este error sucede cuando uno de los armónicos tiene suficiente energía como para producir un máximo local. Esto produce que el tono detectado corresponda a un múltiplo de F0.

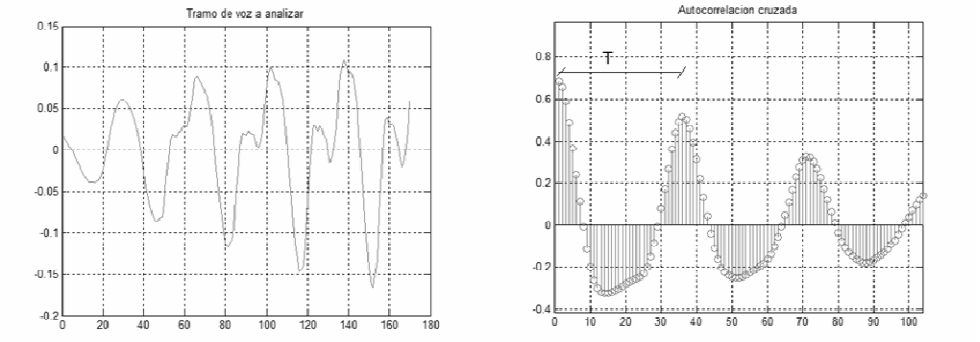

Figura 2: Tramo de señal de voz (izquierda). Parte positiva de la función de autocorrelación (derecha). La distancia entre máximos equivale al periodo fundamental de la señal bajo estudio.]

La autocorrelación ha sido utilizada ampliamente en el ámbito del procesado del habla [15], pero más cercano a la aplicación que nos ocupa son los sistemas de consulta por tarareo (*QbH*) donde [16] o [17] entre muchos otros utilizan la autocorrelación para el seguimiento del *pitch*.

Algoritmo de YIN

El algoritmo de YIN para la estimación de la frecuencia fundamental, es propuesto por Cheveigne and Kawahara [18]. Basándose en la autocorrelación, los autores proponen ciertas variaciones con el fin de paliar los molestos errores de octava, y mejorar los resultados del método.

Dicho algoritmo fue usado satisfactoriamente en la obtención del tono de voz cantada y tarareo por [19], donde tras la evaluación de algunos algoritmos se concluyo que YIN era el más eficiente en la transcripción de estas señales a eventos MIDI.

Dada una señal $s(k)$ muestreada en el dominio del tiempo, con frecuencia de muestreo fs, el algoritmo de YIN estima F0 de la forma:

En lugar de calcular la autocorrelación como el resultado de un producto como en la ecuación (1) , se calcula la diferencia cuadrática $d_t(\tau)$:

$$d_t(\tau) = \sum_{k=t}^{t+W-1} [s(k) - s(k+\tau)]^2 \qquad (2)$$

1. Seguidamente se calcula la media acumulativa normalizada de $d_t(\tau)$, denotada como $d'_t(\tau)$

$$d'_t(\tau) = \begin{cases} 1, & \tau = 1 \\ \dfrac{d_t(\tau)}{\left[\frac{1}{\tau}\right]\sum_{j=1}^{\tau} d_t(j)}, & resto \end{cases} \quad (3)$$

2. Se busca el menor desplazamiento τ con un mínimo local en $d'_t(\tau)$ que sea menor que un umbral dado. Si no existe ningún valor inferior a dicho umbral, se busca el mínimo global de la función $d'_t(\tau)$. Marcándose este desplazamiento como τ'.

3. Para mejorar la precisión se realiza una interpolación polinómica de segundo orden en la función $d'_t(\tau)$, en el rango $[\tau' - 1, \tau' + 1]$.

4. Se busca el mínimo valor de la función interpolada $\hat{\tau}$ en el rango anterior. Siendo la frecuencia fundamental estimada de valor $1/\hat{\tau}$.

El objetivo que se busca en el paso 2 es el de normalizar la función diferencia frente a los cambios de amplitud de $s(k)$, eliminando los picos espurios que puedan aparecer.

Detección en el dominio de la frecuencia

HPS (Harmonic product spectrum)

Toda señal musical armónica está compuesta por frecuencia fundamental, y los armónicos de ésta. Estos armónicos están situados en frecuencias que serán un número entero de veces la fundamental. Si se multiplica el espectro de una señal, por el espectro de esa misma señal interpolada (ecuación 4), el resultado será un aumento de energía en la frecuencia fundamental, y un decaimiento de la misma en los armónicos (ya que al realizar la compresión, los armónicos de alta frecuencia quedarán multiplicados por valores muy pequeños ver Figura 3).

Realizando este proceso varias veces (3 o 4 son suficientes), obtendremos un espectro en el cual solo se podrá apreciar un pico frecuencia correspondiente a la frecuencia fundamental.

$$HPS\ (w) = \prod_{r=1}^{R} |X(w\ r)| \tag{4}$$

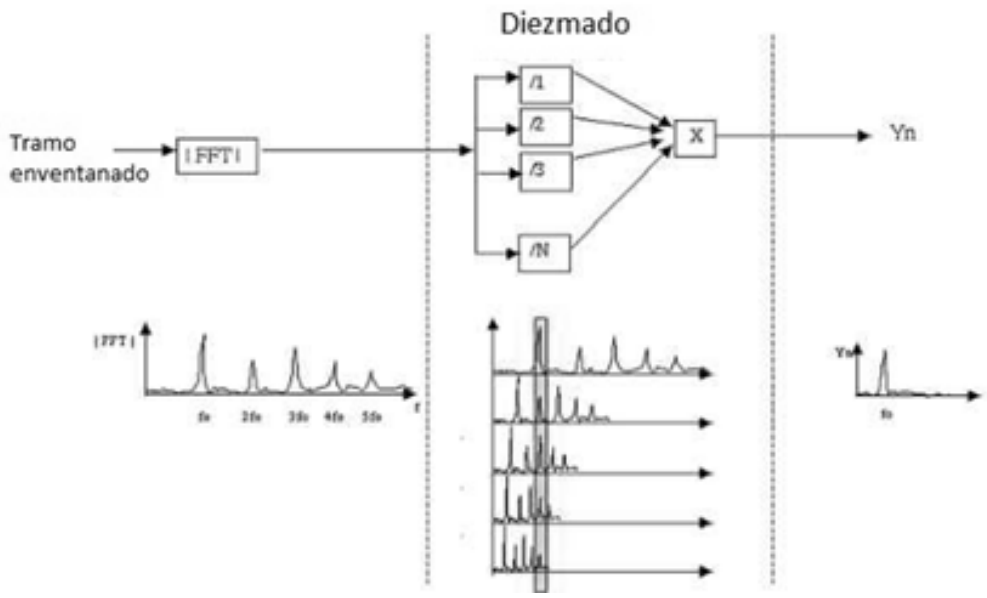

Figura 3: Esquema de procesado del algoritmo HPS para la obtención de F0.

Este método es de gran utilidad para señales fuertemente armónicas como las producidas por cuerda pulsada o frotada.

Su calidad es dependiente de la precisión de la FFT, necesitando ventanas de gran longitud temporal para conseguir alta discriminación frecuencial.

Cepstrum

En el caso particular de la señal de voz, ésta viene dada por la convolucion de la señal de excitación $g[k]$, con la respuesta al impulso del tracto vocal, $h[k]$ y siendo $H(w)$ y $G(w)$, sus respectivas DFT, tendremos que:

$$S(w) = G(w)H(w) \tag{5}$$

Tomando logaritmos del modulo de la ecuación (5) conseguimos linearizar el sistema frecuencial :

$$log\,|S(w)| = log|G(w)| + log|H(w)| \quad (6)$$

Si además se calcula la DFT-1, obtenemos:

$$c(\tau) = DFT^{-1}log|S(w)| = DFT^{-1}log|G(w)| + DFT^{-1}log|H(w)| \quad (7)$$

En este dominio, llamado cepstral (resultado de invertir "spec" en la palabra "espectral"), las componentes debidas a la envolvente y las variaciones finas del espectro aparecen como sumandos, por lo que pueden ser separados fácilmente mediante un filtrado lineal (llamado liftrado en este dominio).

Las componentes debidas a los armónicos aparecen como picos equiespaciados en altas quefrencias (ver Figura 4), esto permite obtener el periodo fundamental, y por ende la frecuencia fundamental, cuyo objetivo se persigue.

Estos predictores de *pitch* son ampliamente utilizados en el procesado del habla, pero el hecho de tomar logaritmos produce un aumento del ruido de baja frecuencia, lo que los hace sensibles a error en ambientes ruidosos y señales captadas pobremente. Por otra parte, los algoritmos basados en cepstrum son muy precisos.

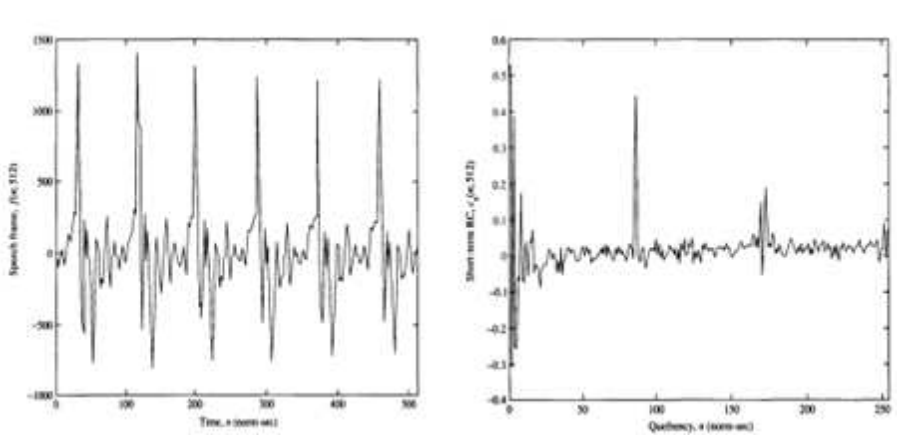

Figura 4: Tramo sonoro de voz (izquierda) y cepstrum (derecha)

2.2.3 Segmentación de eventos

En la transcripción de la voz, es necesaria la determinación del inicio y el final del evento denominado nota. Esto es usualmente definido por dos tipos de características: aquellas que

determinan la existencia de un silencio, ruido o tramo sordo, y aquellas que determinan el inicio o fin de una nota.

- Energía: Las características debidas a las variaciones energéticas de la señal son ampliamente utilizadas para segmentación. Se asume que un evento de interés posee mayor energía. Estableciéndose un umbral, se puede distinguir entre los tramos de voz y el ruido o señales de fondo.

  Una de las medidas de energía más usadas es el valor cuadrático medio (RMS *Root Mean Square*). los previamente mencionados sistemas [3], [4] y [9] hacen uso de éste para realizar el segmentado. Y diferentes cálculos basados en la energía utiliza como herramienta [2].

- Sonoro/sordo: Una característica más robusta, es la observación de la sonoridad del tramo, pudiendo ser éste sonoro o sordo. Evaluando el nivel de periodicidad del tramo, o calculando el valor de la tasa de cruces por cero, se puede realizar una distinción robusta.

- Acentos: Un indicador del inicio de una nota viene dado por el acento imbricado en ésta. Entendiendo acento como el momento en el que se tiene la percepción de un énfasis en la señal musical, éstos pueden ser utilizados para determinar los inicios de evento.

  En [20] se propone un método de detección basado en ésta característica. El proceso es el siguiente. Se obtiene la posición temporal del acento en base al aumento de energía en el espectro.

  En primer lugar, la señal se pasa por un banco de 36 filtros. Se calcula la energía de cada banda, y ésta es comprimida logarítmicamente. La envolvente de la energía es calculada para obtener la derivada y así poder evaluar los cambios de intensidad. Esta señal de acentos es umbralizada resultando los candidatos a inicio de nota.

Finalmente, la información de cada banda es combinada para obtener la segmentación de eventos.

2.2.4 Pos procesado y codificación

La etapa de pos procesado es utilizada para la codificación de la información extraída de la señal de audio, frecuencia fundamental y comienzo y fin del evento. Pero también pueden realizarse mejoras en la interpretación original.

Una de estas mejoras es la cuantización de tiempo, la cual consiste en asignar los inicios de nota a valores discretos de una rejilla temporal, esto está basado en la métrica musical, todo evento que comience en un instante que no corresponda a la rejilla, se considerará error en la interpretación y se desplazará consiguiendo una mayor precisión. Para realizar una buena cuantización, es aconsejable la utilización de un algoritmo de seguimiento de tempo, para así establecer el espacio mínimo de la rejilla.

Como ya se ha comentado, otra fase del pos procesado es la codificación del evento. Pero en un tramo de voz existen más matices además de la nota que se está interpretando, cuando empieza y cuando finaliza, nos referimos a los antes mencionados *vibrato glisando o tremolo* entre otros.

Estas variaciones en frecuencia pueden ser medidas mediante un algoritmo de búsqueda de picos (*peak-picking*) en las variaciones de tono a lo largo de la nota [21], para posteriormente ser codificadas como mensajes de *pitch bend*.

De la misma forma, las variaciones de dinámica, pueden medirse como diferencias en el valor RMS de la señal, y codificarse mediante *aftertouch*.

Capítulo 3

Transcripción rítmica

3.1 Descripción del problema

La transcripción rítmica generalmente tiene el objetivo de extraer de la interpretación de una batería o de elementos percusivos, y transcribirlas a notación musical como pueda ser un pentagrama o a mensajes MIDI.

El proceso de la transcripción musical está caracterizada por dos preguntas: ¿Cuándo ha sucedido un evento? y ¿A que se corresponde ese evento?. En la transcripción rítmica, esas dos cuestiones se dan en el mismo orden.

- Segmentar la señal de entrada
- localizando los potenciales inicios de eventos, o
- generando una rejilla temporal sobre la señal .
- Extraer características de cada segmento.
- Clasificar el contenido de cada segmento basándose en las características obtenidas.
- Combinar la información temporal del evento con la clasificación realizada, para completar la transcripción.

3.1.1 Localización de comienzo *(onset)*

Los sonidos percusivos están generalmente caracterizados por un significativo incremento de energía en la señal (transitorio).

Recientes estudios como en [22] se trata de localizar los eventos relevantes en el habla, teniendo en cuenta el incremento de energía y los cambios espectrales calculados por la FFT además de otras características del espectro. Es interesante para observar la variedad de sistemas en detección de eventos, pero la voz difiere en muchos aspectos de las señales percusivas.

Por otra parte, Collins [23] desarrolla un algoritmo de detección de señales percusivas desprovistas de tono basándose en la toma de ventanas y obtención de la FFT. También localiza los súbitos incrementos de energía y fortalece la detección incluyendo la tasa de cruces por cero.

En este tipo de detección se suele trabajar sobre la envolvente, calculada con un enventanado con solapamiento de 50 o 75% y de aproximadamente 20ms [24]. Este también abre la posibilidad de usar diferentes bandas, de 20-200Hz por ejemplo, es útil en la separación de eventos, de 200 a 15kHz es donde se encuentra la mayor parte de los armónicos, y de 15kHz en adelante, los armónicos han desaparecido pero contiene información sobre los transitorios. También se propone hacer uso de la escala de Mel u otros criterios psicoacústicos.

Rejilla Temporal

Una alternativa a la detección de onset es la creación de una rejilla temporal a lo largo de toda la señal y usarla para su segmentación [25]. La separación está determinada por la separación mínima entre eventos que exista, de esta forma todos los eventos contenidos en la señal se coincidirán con la rejilla (teniendo en cuenta una buena interpretación).

En la práctica, hacer uso de de una rejilla no ofrece buenos resultados, ya que depende en gran medida de la detección de tempo. Además, esta técnica excluye la posibilidad de hacer uso de este método en tiempo real, ya que es necesario conocer el tempo de la señal al completo.

Una gran ventaja es que mejora la detección, eludiendo posibles tramos de señal que no correspondan a los puntos marcados por la rejilla.

3.1.2 Extracción de características

Ya teniendo marcados los eventos, se procede a la extracción de características. Como se propone en [24] es útil definir un tamaño mínimo y máximo para cada tramo obtenido. Valores típicos son de 50 a 200ms.

También se puede hacer uso de una función de enventanado teniendo presente que las comunes Hamming o Hanning tienen el defecto de suavizar el ataque de la señal. En su lugar [24] propone utilizar una ventana mitad Hanning que comience en el valor unidad y decaiga hasta el final del segmento. Pero en ocasiones el enventanado es omitido totalmente, ya que se presupone una caída natural de la energía de la señal dado que ésta no tiene sostenimiento en absoluto.

El objetivo de la extracción de características, es el de obtener una serie de valores que describa los segmentos reduciendo de esta forma la gran cantidad de información temporal, y poder clasificarlos más eficientemente. Se han realizado investigaciones sobre las características más descriptivas de los sonidos percusivos.

En [26] se aborda la importancia de la tasa de cruces por cero del ataque de la señal como diferenciador de fuente. En [8] se realiza un estudio de las características que mejor discriminan los elementos de un set clásico de batería, teniendo en cuenta tanto características temporales como espectrales.

Se van a describir las características más usadas, idealmente, éstas deben ser suficientemente robustas como para mantenerse en presencia de ruido.

Los coeficientes cepstrales de Mel (MFCCs), permiten obtener una descripción del espectro, sin necesidad de tener toda la información contenida en éste. Usualmente los coeficientes son calculados en cortos periodos de señal, típicamente 20ms [27] , con solapamiento parcial, obteniéndose de 5 a 15 coeficientes. En vez de utilizar estos valores directamente, es usual utilizar la media y varianza de cada coeficiente a lo largo del segmento.

Otro grupo de características espectrales que determina a grandes rasgos la forma del espectro, son los derivados de los órdenes de los momentos del espectro, como son: el centroide, la dispersión, la planitud y la curtosis. También puede hacerse uso de estos descriptores tomando eje logarítmico en vez de lineal como se propone en la norma MPEG-7 [28].

Además de las cualidades espectrales previamente mencionadas, es común hacer uso de las características temporales de la señal. Estos pueden ser la tasa de cruces por cero y análogamente con el espectro, el centroide temporal de la envolvente.

Generalmente, el set de características es testado mediante el método de prueba y error. Pero es aconsejable hacer uso de un sistema automático de selección de características. Este tipo de sistemas es evaluado por Herrera [8]. Por lo general, hacer uso de un sistema automático de selección de características tiene mejores resultados. Además, un sistema de reducción dimensional como el análisis por componentes principales puede ser utilizado para elegir los descriptores más adecuados, previo entrenamiento.

Selección de características

A la hora de distinguir los diferentes sonidos en base a una gran cantidad de características, lo más común es que algunas sean de mayor utilidad que otras. Pero la utilidad de las mismas variara dependiendo de la situación.

Imaginemos por un momento que se está desarrollando un sistema de reconocimiento personal que clasifica a cada persona por su país de origen en Europa, y las características que se proponen son: altura, color de ojos y color de pelo. Las diferencias entre un noruego de gran envergadura, rubio y ojos azules, con un italiano de menor estatura, moreno y con ojos negros son más que suficientes como para realizar una buena clasificación. Ahora bien, si ese mismo sistema se prueba en Asia, los resultados serán pobres, ya que ni el color de pelo, ni los ojos, ni siquiera la altura suponen unos buenos descriptores. Por lo que sería de gran utilidad un algoritmo que decidiera previa verificación qué características son más decisivas.

La evaluación de una característica debe tener en cuenta que el valor de esta sea diferente para cada clase (interclase) y que se mantenga lo más constante posible en cada una de ellas (intraclase).

En [8] se realiza un estudio sobre las técnicas de selección de características para instrumentos percusivos sin tono. En él se evalúan diferentes algoritmos de selección como son el del cálculo de *los K vecinos más cercanos* (KN-N), o el *análisis canónico de discriminación* más conocido como algoritmo ANOVA o MANOVA. Este último hace uso de un modelado estadístico que discrimina las clases en base al centroide de la distribución de probabilidad.

3.2 Redes neuronales artificiales: fundamentos.

3.2.1 Introducción

El proceso de computación tradicional es determinista, secuencial y lógico. Esto nos permite realizar operaciones matemáticas a una gran velocidad y eficiencia. A un ser humano no se le puede pedir la resolución de una ecuación diferencial en milésimas de segundo, algo que es realmente sencillo para un ordenador.

Pero en cambio, nuestro cerebro es capaz de diferenciar infinidad de caras, objetos, sonidos e incluso formular hipótesis que predigan la evolución de las situaciones. Tarea muy difícil para un ordenador ¿por qué?, por la estructura jerárquica y en paralelo de elementos de procesado llamados neuronas que posee nuestro cerebro.

Las redes neuronales artificiales pretenden resolver aquellos problemas que un ordenador tradicional no puede. Estas han sido utilizadas en numerosas áreas. Redes con menos de 150 elementos se han testado satisfactoriamente en el control de vehículos [29], síntesis de habla [30], y detección de explosivos en el equipaje de pasajeros de avión [31] entre otras muchas aplicaciones.

El uso de redes neuronales ofrece la posibilidad de disponer de una poderosa herramienta cuyas propiedades incluyen:

- **Aprendizaje**: Pueden ser entrenadas para asociar patrones entrada-salida.

- **Generalización**: No solo memorizan los datos que se usan en el aprendizaje; éstas pueden generalizar lo aprendido a nuevos ejemplos.
- **No-linealidad**: pueden procesar funciones no lineales, permitiendo la interpretación de cualquier transformación arbitraria de los datos de entrada.
- **Robustez**: Son muy tolerantes al ruido, es más, dicho ruido puede ayudarlas a realizar mayores generalizaciones.

3.2.2 Historia

La historia de las redes neuronales artificiales comienza con los primeros estudios del cerebro humano. Pero son los avances en la electrónica los que han permitido intentar emular un cerebro humano así como su forma de pensar y razonar.

Walter Pitts y Warren McCullow realizaron una aproximación de lo que ellos pensaban era el funcionamiento del cerebro. Mediante una red de elementos de cálculo, que podían realizar operaciones lógicas. Cada "neurona" tenía dos estados activada/desactivada.

En 1949 Donald Hebb [32], expone la regla de aprendizaje que se conocerá como hebbiana o de Hebbian. Basándose en que la repetida activación de una sinapsis aumenta la conectividad de la unión, y por ende la hace más propensa a ser usada sucesivamente. La regla dicta:

``When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased'' [1]

El primer experimento se dio en 1951, cuando Minsky y Edmonds crearon una red neuronal de 40 neuronas. El objetivo era testar la red en la resolución de un laberinto de 40 celdas (cada celda representaba una neurona). La neurona activa indicaba la posición en la que se encontraba, y la decisión de la siguiente celda que ocupar estaba basada en la regla de

[1] Cuando un axón de una célula A está lo suficientemente cerca de una célula B como para excitarla, y participa repetida o persistentemente en su activación, ocurre algún proceso de crecimiento o cambio metabólico en una o en ambas células, de modo tal que la eficacia de A, como una de las células que hacen activar a B, aumenta.

Hebbian, por lo que la sinapsis que había sido más usada (el sistema había recorrido el laberinto con anterioridad) era la elegida.

En 1957, Frank Rosenblatt presentó el Perceptrón, introduciendo el aprendizaje supervisado (ver explicación en pagina). El dispositivo de entrada consistía en una señal binaria cuyas conexiones a la red eran ponderadas y combinadas. Si el resultado superaba un umbral dado, la neurona se activaba. Por cada ciclo la salida obtenida era comparada con la salida deseada, y aquellas conexiones que daban mejores resultados tenían como resultado un aumento de la ponderación (peso de la conexión).
El perceptrón al que llamo Mark I, consistía en una matriz de 20x20 células fotoeléctricas conectadas a una red eléctrica con conexiones variables.

Bernard Widrow y Marcian Hoff desarrollan en 1959 los modelos ADALINE y MADALINE (*Multiple ADAptive LINear Elements*). ADALINE fue entrenada en el reconocimiento de los patrones binarios de una línea de teléfono, y podía predecir el siguiente bit de la cadena. MADALINE fue la primera red usada en la resolución de un problema real: eliminar los ecos en una línea telefónica, mediante un filtrado adaptativo.

A pesar de esos avances, en los años 60 llega la arquitectura tradicional de computación von Neumman, cuyo auge deja de lado la investigación de las redes neuronales artificiales.

En la actualidad, las redes neuronales tienen múltiples aplicaciones. Desde la transcripción de texto manuscrito, la síntesis del habla, o la predicción del mercado de valores a la detección de minas submarinas y sistemas de control para automóviles.

La versatilidad de estos sistemas no es comparable a la capacidad del cerebro humano, pero es inevitable que aquellos con un pequeño papel en el mundo de la tecnología se pregunten sobre la inteligencia artificial y los avances en este área. ¿Se conseguirán crear redes neuronales capaces de resolver aquellos problemas que un cerebro humano no puede ni siquiera imaginar?. El autor de este proyecto así lo opina.

3.2.3 Modelos biológico y artificial

El cerebro humano está compuesto por aproximadamente 10^{11} neuronas con una gran cantidad de interconexiones 10^4 aproximadamente por elemento.

Las dendritas constituyen el receptor eléctrico que lleva la señal al cuerpo o núcleo de la neurona (ver Figura 5). Este cuerpo suma las señales de entrada y decide la salida que va a

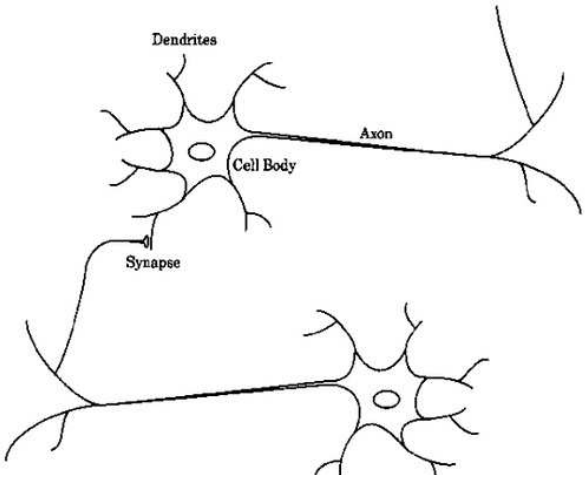

Figura 5: modelo esquemático neuronal biológico.

proveer a las demás neuronas a las que está conectado, en primer lugar mediante la sinapsis, y posteriormente a través del axón.

Las redes neuronales artificiales poseen ciertas similitudes con las biológicas. Al igual que en el cerebro humano, la red artificial se construye con unidades sencillas de procesado (neuronas) provistas de un gran número de interconexiones entre sí, cuya salida vendrá dada por una función de transferencia.

Modelo de neurona artificial

Como se comentaba en el modelo del perceptrón, cada conexión sináptica tiene un peso (una ponderación), cuyo valor será determinado por el aprendizaje al que haya estado sometido.

Además del peso, al resultado de la combinación de todas las conexiones que lleguen a la neurona, se le suma un valor (bias). Esto permite que la salida de la neurona tenga mayor flexibilidad de comportamiento.

El modelo de neurona artificial se muestra en la Figura 6. Siendo p el valor escalar de entrada, el cual es multiplicado por el peso w que como se ha comentado habrá sido asignado previamente en la creación de la red o en la fase de entrenamiento. Este producto $\boldsymbol{pw}$ es sumado con el umbral o bias $\boldsymbol{b}$ creando la entrada a la función de activación o transferencia. Cabe destacar que en ocasiones no se realiza una suma, sino un multiplicación por ejemplo; pero es la suma la operación más común.

Finalmente obtenemos la salida de la neurona como:

$$a = f(wp + b) \tag{8}$$

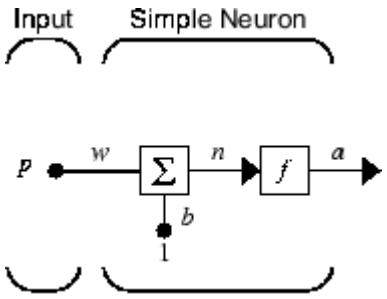

Figura 6: Modelo de neurona artificial.

La idea a tener clara es que se dispone de una unidad de procesado cuyas variables a configurar son peso y bias, estos dos parámetros serán ajustados para que la neurona tenga el comportamiento deseado.

Función de transferencia o activación

Es la función que determina el valor final de la neurona. Existen gran variedad de funciones, pero se pasa a discutir las más usadas.

- Función Escalón: Se muestra en la Figura 7 (izquierda), ésta da como resultado a su salida 0, si su entrada es menor que 0, y 1 en caso contrario. Esta función es de utilidad a la hora de clasificar una serie entradas en dos diferentes clases. En la Figura 7 (derecha) podemos observar la respuesta total de una neurona que hace uso de la función escalón.

Se puede observar gráficamente como el peso y el bias asignado varían la salida de la neurona modificando el umbral de activación de ésta.

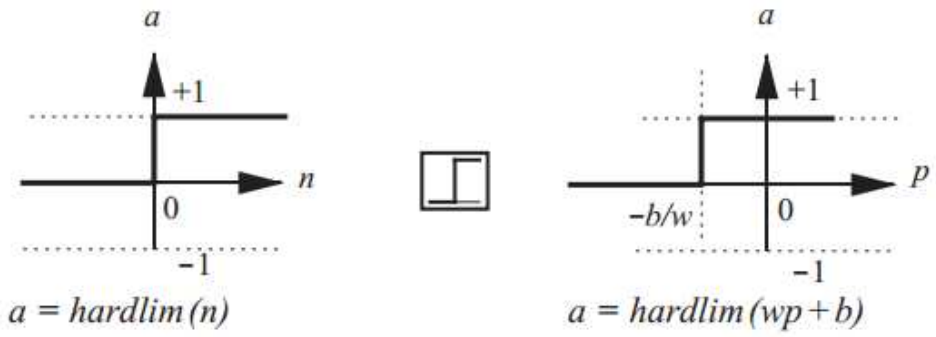

Figura 7: Función de activación escalón. Sin efecto de bias (izquierda), y con él (derecha). Bajo ellas se detalla la ecuación de salida.

- Función Lineal: dicha función asigna el mismo valor de entrada a la salida, solo variándose por el bias. En este caso, la salida puede tomar cualquier valor; esto la hace más útil en problemas que requieran la distinción de más de dos clases. Es de utilidad como función de activación para neuronas que se encuentran en la capa de salida y no en una capa oculta.

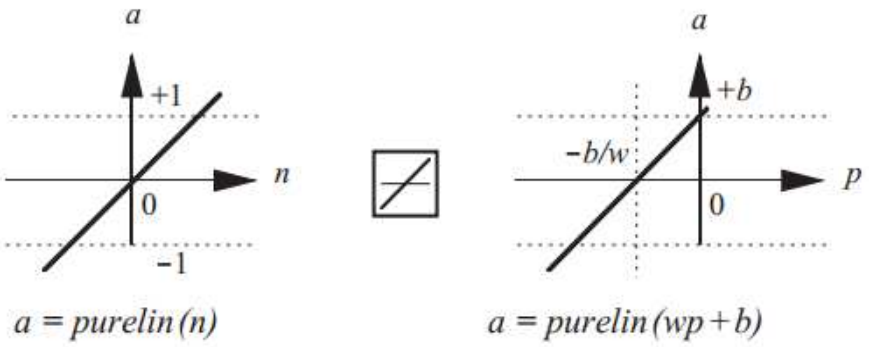

Figura 8:Función de activación lineal. Sin efecto de bias (izquierda), y con él (derecha). Bajo ellas se detalla la ecuación de salida.

- Función Sigmoidal: Esta función toma el valor de entrada entre $-\infty$ y $+\infty$ y limita su salida entre 0 y 1. Esta función es muy usada en las capas ocultas. Esta función de transferencia se muestra en la Figura 9.

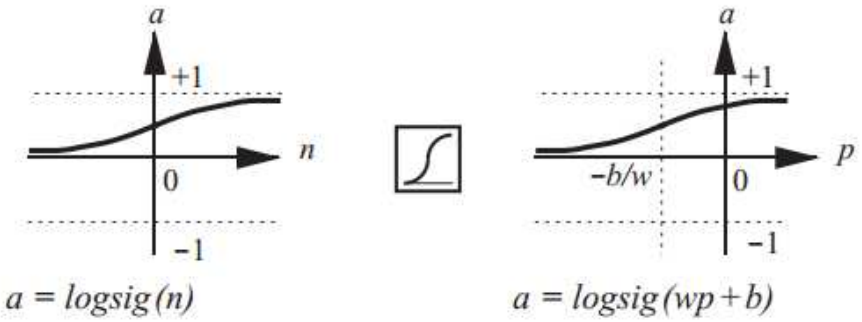

Figura 9: Función de activación log-sigmoidea. Sin efecto de bias (izquierda), y con él (derecha). Bajo ellas se detalla la ecuación de salida.

En numerosas ocasiones, interesara que la salida de la función de transferencia tome el rango [-1,1], de esta se consigue un rango más amplio para la posterior clasificación, para lo cual se define lo función tan-sigmoidal.

Para una mayor profundidad sobre funciones de transferencia ver [33].

3.2.4 Arquitecturas

La sencilla unidad de procesado llamada neurona es generalmente, combinada en paralelo para crear lo que se conoce como ***capa*** mostrado en la Figura 10 (izquierda). Nótese que cada valor escalar de entrada ***p*** está conectada a cada neurona de la capa, por lo que cada uno viaja por toda la red.

En la Figura 10 (derecha) se muestra la notación abreviada que se utilizara en adelante, siendo ***S*** el número de neuronas de la capa, ***R*** el número de valores del vector de entrada y ***f*** el conjunto de ***S*** funciones de transferencia, de idéntica naturaleza.

En una red de varias capas, la salida de la primera constituya el vector de valores de entrada para la siguiente y así en adelante.

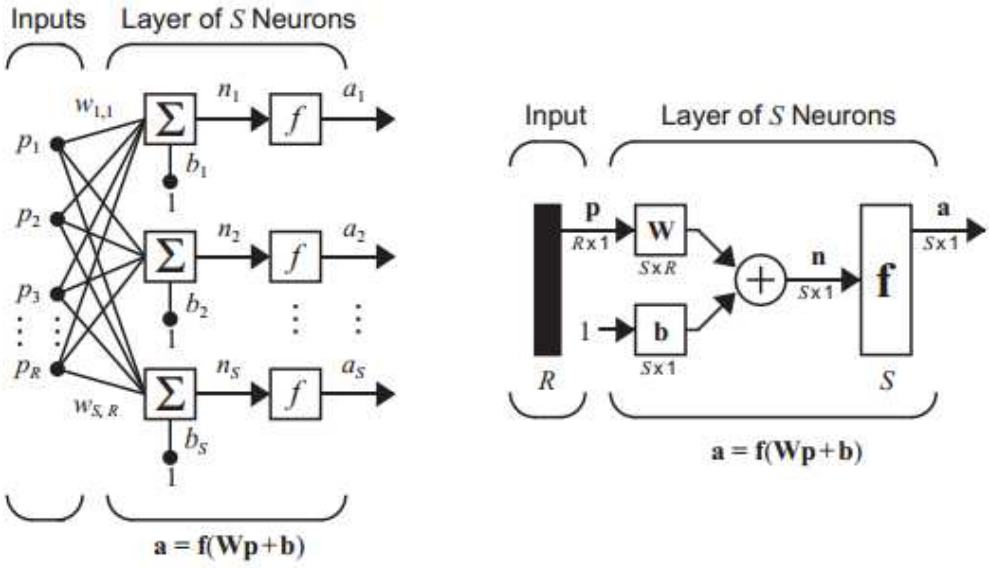

Figura 10: Esquema del flujo de información en una red neuronal (izquierda), y su notación abreviada (derecha).

Haciendo uso de varias capas, creamos una estructura jerarquizada mucho más potente que la de una única capa. Podría pensarse que a mayor número de capas, más inteligente es la red, y esto es cierto hasta cierto punto. El aumento de coste computacional y la escasa mejora (o empeoramiento) tiene como resultado que la gran mayoría de las redes artificiales tengan tres o cuatro capas ocultas (las capas que se encuentran entre las entradas y la capa de salida) como máximo.

A este tipo de redes en las cuales la información fluye desde la capa de entrada hasta la salida se les suele conocer como redes estáticas (*feed forward*).

Redes neuronales dinámicas o recurrentes

A diferencia de las redes estáticas cuyo flujo de información es "hacia adelante" y la salida de la red es calculada únicamente en base a las entradas. Las redes dinámicas o recurrentes poseen uno o varios lazos de realimentación, teniendo como resultado sistemas con memoria cuya salida se calcula en base a las entradas y a las muestras retardadas que provendrán del lazo de realimentación.

Este tipo de redes pueden ser entrenadas para reconocer secuencias. Son usadas en reconocimiento de hablante, para predecir el comportamiento de mercados y ecualización de canales de comunicaciones entre muchas otras aplicaciones [34].

Un ejemplo genérico de una neurona recurrente se muestra en la Figura 11. En ésta se puede observar como la entrada $\boldsymbol{p}$ es sumada con la muestra que ya ha recorrido la red en el instante anterior. Por lo que la salida $\boldsymbol{a}$ resulta:

$$a = iw_{11}p(t) + Iw_{11}a(t-1) \tag{9}$$

En la ecuación (9) se ha obviado la influencia de la función de transferencia, ya que por pura simplicidad se ha elegido la función lineal.

Existen numerosas topologías de redes neuronales recurrentes. Se pasa a describir una de las más sencillas y comunes, la red de Elman (Figura 12):

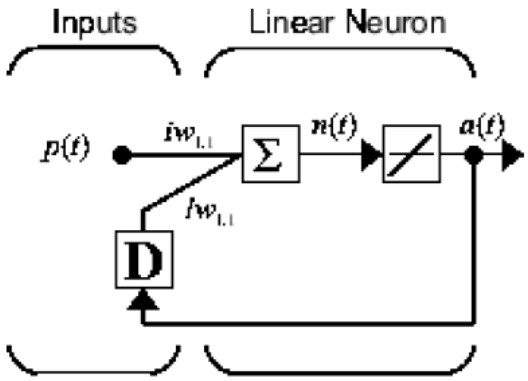

Figura 11: Modelo de neurona artificial con lazo de realimentación.

La red original cuenta con dos capas, usando la función tan-sigmoidal para la activación de la capa oculta, y la lineal para la capa de salida [35]. La realimentación se realiza de la salida a la entrada de la capa oculta.

Dependiendo de la red que se desee implementar, estas realimentaciones pueden mantenerse en una capa o unir distintas.

Como podemos observar, la realimentación también es "ponderada" por un peso, siendo tratada como otra entrada más a la red.

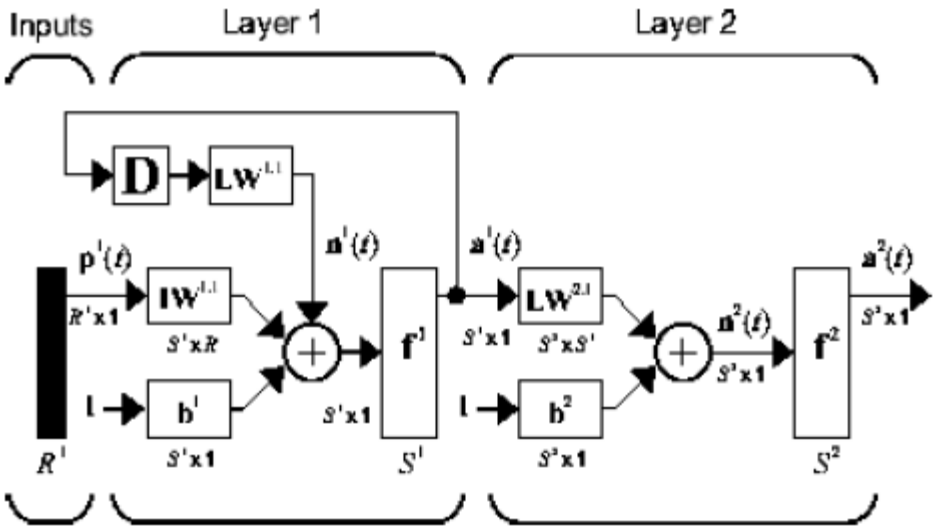

Figura 12: Diagrama en notación abreviada de una red de Elman.

Las combinaciones de redes neuronales son prácticamente infinitas. Incluso pueden combinarse redes de propagación hacia adelante con recurrentes, teniendo capas con memoria y capas sin ella. Por no decir las posibles conexiones de realimentación entre los diferentes niveles.

Usualmente, se utiliza un tipo de función de transferencia para todas las neuronas de una misma capa, pero también puede elegirse una específica para cada neurona, ampliando las posibilidades en el diseño.

3.2.5 Fase de aprendizaje o entrenamiento

Al igual que a un niño hay que enseñarle a escribir antes de pedirle la redacción resumen de su verano, una red neuronal artificial debe ser sometida a un proceso de aprendizaje antes de ser utilizada.

El encargado de entrenar a la red es el llamado algoritmo de entrenamiento. El objetivo de dichos algoritmos es ajustar los pesos y bias de cada elemento para que la red implemente

la salida deseada. Uno de los criterios más comunes es el de minimizar el error de salida (diferencia entre la salida obtenida y la salida deseada).

Este entrenamiento se realiza mediante un proceso iterativo, motivo por el que existen dos maneras de aplicar estos algoritmos. El método *on-line* (*incremental*) en el cual los pesos se varían cada vez que se introduce una entrada en la red, y off-line (*batch mode*) en el cual la modificación se realiza una vez que se han introducido todas las entradas.

Ya sea *on-line* u *off-line,* se distinguen varias formas de entrenar una red neuronal, las cuales guardan asombrosas similitudes con el aprendizaje humano. Algunas ya han sido introducidas brevemente.

Aprendizaje supervisado

En este entrenamiento, la red neuronal asiste a la escuela con un profesor que le indica en qué medida está haciéndolo bien o mal. De esta forma, puede variar la forma en la que se comporta para mejorar su trabajo.

Se le pasan las variables de entrada, y lo que debe ofrecer a su salida. En base al error cometido, se van modificando los pesos y bias de cada neurona para que el resultado sea lo más parecido al objetivo perseguido.

Un ejemplo en la vida de un niño podría ser el de darle a las piezas de un puzle, y la caja en la cual aparece el dibujo completo. Poco a poco irá viendo como unir las piezas para componer el resultado correcto.

Propagación hacia atrás (Back-propagation)

Los algoritmos de entrenamiento son una potente herramienta pero de alto coste computacional. Una red *feed-forward* con una capa oculta entrenado con el algoritmo de aprendizaje supervisado conocido como de *propagación hacia atrás* [36] puede modelar cualquier función continua con el grado de precisión que se desee (variando el número de neuronas en la capa oculta). El proceso de aprendizaje es el siguiente:

Dada una red neuronal con topología hacia delante (*feed-forward*). A ésta se le presenta una entrada; la cual va siendo modificada por los pesos y bias en su recorrido por el "axón" artificial, y evaluada por la función de transferencia de cada neurona hasta obtener un valor en

su capa de salida, dicha salida puede ser cualquier número, recordemos que los pesos y bias pueden poseer un entrenamiento inicial o pueden ser hasta números aleatorios.

Al ser un método de entrenamiento supervisado se sabe cuál debería ser la salida, por lo que es posible obtener la diferencia entre la salida obtenida y la deseada.

Este error se utiliza para variar los pesos de la red minimizando la diferencia entre la salida deseada y la obtenida.

Cada peso se modifica en base al factor de aprendizaje, el cual indica la velocidad a la que se varían los parámetros para minimizar el error. A mayor factor de aprendizaje, mayor es la velocidad con la que la red aprende. Hay que tener cuidado con este parámetro ya que afecta a la estabilidad del sistema.

Tras modificar los pesos desde la capa de salida, a través de la(s) capa(s) ocultas. Se evalúa si el error ha disminuido o no.

Este algoritmo presenta ciertos problemas. Dado que se busca minimizar el error en todo instante, este puede aumentar ligeramente para luego decaer hacia un valor menor aun. Pero en este caso el algoritmo se quedaría "atascado" en el mínimo local, sin llegar al mínimo absoluto.

La solución a este problema es introducir *momentum* a la variación de los pesos, es decir, que la variación del peso en un instante no solo dependa de ese error, sino también del anterior.

Este algoritmo ha sido utilizado en multitud de sistemas. Algunos que se remontan a finales de los años 80 como es el sistema NETtalk [22], el cual aprendía como pronunciar texto en inglés. O el también antiguo Neurogammon [37], el cual se entreno para diseñar estrategias en el famoso juego Backgammon . Una aplicación más sorprendente es ALVINN, red neuronal entrenada en la conducción de un coche; los datos de entrada se correspondían a los obtenidos por una videocámara que apuntaba a la carretera, el sistema aprendía de cómo conducía una persona.

Algoritmo CGB. Conjugate Gradient with Powell/Beale Restarts

Sobre el algoritmo descrito previamente, se han propuesto variaciones, que puedan mejorar la eficiencia del entrenamiento.

Minimizar la función de error podría asemejarse a intentar encontrar el menor punto de una superficie caminando sobre ella.

En el algoritmo CGB el valor de pesos y bias se elige en el valor más pronunciado de la función de error, (cuando el gradiente de dicha función es mínimo). Esta es la dirección en la que dicha función decrece más rápidamente.

En los algoritmos de gradiente conjugado, la minimización del error se realiza en direcciones conjugadas, es decir, caminando por esa superficie solo se podría girar ± 90 grados. Luego primero se busca el menor gradiente negativo en una dirección y después se busca el menor gradiente negativo en la dirección conjugada.

Esto se realiza iterativamente, encontrando el valor general que reduce la función de error en todas direcciones.

Ahora bien, lo que se ha explicado anteriormente es general para cualquier método basado en el gradiente conjugado. Para éstos algoritmos la dirección de búsqueda es reiniciada periódicamente al gradiente negativo, pero existe otra forma de reinicializar la dirección, propuesta por Powel [38], basado en una versión anterior de Beale [39]. Esta modificación dicta que se cambie de dirección si existe cierta ortogonalidad entre el gradiente y su predecesor. Matemáticamente, se inicializará la búsqueda si se cumple:

$$\left|g^{T}{}_{k-1} g_k\right| \geq 0.2\|g_k\|^2 \qquad (10)$$

siendo g el gradiente y k el instante presente. El valor 0.2 representa la "cierta ortogonalidad" de la que se hablaba, en el caso propuesto por Powel, un 20% de ortogonalidad sería suficiente para cambiar de dirección de búsqueda.

Algoritmo LM. Levenberg-Marquardt

Este algoritmo de Aprendizaje supervisado, es usado ampliamente en el entrenamiento de todo tipo de redes ya sean estáticas o dinámicas. Se describe sucintamente a continuación.

Teniendo Ecomo la funcion de error a minimizar y situándonos en la interacción $x[n]$, el objetivo es encontrar la siguiente interacción $x[n+1]$ que cumpla $E(x[n+1]) < E(x[n])$ es decir, que el error en la siguiente iteración sea menor que en la actual. Para elegir $x[n+1]$ es necesario saber la direccion hacia la que vamos a desplazarnos y el tamaño del paso (cuán lejos deseamos ir).

Este algoritmo, es un escalón intermedio entre el método del gradiente conjugado explicado anteriormente y el algoritmo Newton-Gauss (GNA).

La ecuación general de Newton es la siguiente

$$\boldsymbol{N}\Delta = \boldsymbol{J}^{\boldsymbol{T}}\boldsymbol{J}\Delta = \boldsymbol{J}^{\boldsymbol{T}}\boldsymbol{\varepsilon} \tag{11}$$

donde $\boldsymbol{J}$ representa el jacobiano de la funcion, Δ el incremento de los paramentros (en nuestro caso, los pesos y bias) y $\boldsymbol{\varepsilon}$ los errores del ajuste. Esta es sustituida por la ecuación modificada

$$\boldsymbol{N}'\Delta = \boldsymbol{J}^{\boldsymbol{T}}\boldsymbol{\varepsilon} \tag{12}$$

donde $\boldsymbol{N}' = (\mathbf{1} + \boldsymbol{\lambda})$ siendo $\boldsymbol{\lambda}$ un parámetro que define la longitud del avance.

Si resolviendo las ecuaciones el valor de Δ aumenta, entonces el incremento es tomado como incorrecto y los pesos y bias se ajustan a la anterior iteración, $\boldsymbol{\lambda}$ es aumentado multiplicándose por un valor (10 normalmente).

En caso de que el error se haya minimizado, los pesos y bias son actualizados y se decrementa $\boldsymbol{\lambda}$.

Este algoritmo es el más usado en las tareas de discriminación de patrones sonoros. En [40], se hace uso de una red *feed-forward* de dos capas ocultas entrenada con LM, ofreciendo una precisión incluso superior al 99% en la clasificación de señales de voz. LM ofrece una buena relación entre tiempo invertido en el entrenamiento, y error a su salida [41], claro está, esto depende de la aplicación en la que se esté trabajando.

En [41] se implementa un sistema de reducción de ruido, en donde se testan redes tanto dinámicas como estáticas, resultando LM el algoritmo más eficiente.

Aprendizaje por refuerzo

Es similar al aprendizaje supervisado, pero la diferencia es que a la red no se le dice exactamente qué debe obtener a su salida, simplemente se le da una "nota" que refleja lo bien o mal que ha realizado la tarea. Y en base a ese valor habrá que cambiar en mayor o menor medida los pesos y bias.

Aprendizaje no supervisado

A la red se le pasan las entradas, pero no las salidas que deben resultar. El propio sistema debe decidir cómo adaptarse. Sería como darle al niño las piezas del puzle y dejarle que en base a los errores que va cometiendo aprenda a construirlo. Esto permite a la red neuronal enfrentarse a problemas nuevos y recorrer el camino para solucionarlo. Es como un cerebro humano se enfrenta a nuevas situaciones y problemas.

Este tipo de entrenamiento es el perseguido por la inteligencia artificial, y la robótica. Pero en la práctica, el aprendizaje supervisado o por refuerzo ofrecen mejores resultados. Pero no deja de ser una línea de investigación de gran interés y potencial en el mundo de la tecnología.

Aprendizaje de Hebbian

Según Hebbian [24], si dos neuronas que se encuentran conectadas son activadas simultáneamente, la conexión entre ellas debe ser fortalecida (traducida como un incremento en el peso de la conexión). Por lo que teniendo dos neuronas conectadas *i* y *j* con sus respectivas entradas x_i y x_j, el peso de la conexión w_{ij} segun el aprendizaje hebbiano vendria dado por la expresión:

$$w_{ij} = x_i \cdot x_j \tag{13}$$

puede existir otras muchas expresiones que describan el fortalecimiento de la conexión (peso) en este tipo de aprendizaje, siendo ésta la más sencilla.

División de los datos

A la hora de realizar cualquier entrenamiento, generalmente se dispone de un número finito de datos. El primer paso es dividir los datos en diferentes grupos. El primero es el llamado grupo de entrenamiento, el cual se utiliza para calcular el gradiente (en el caso de algoritmos basados en éste) y configurar los pesos y bias de la red. El segundo es el grupo de

validación, monitorizándose el error durante este proceso. Usualmente, el error disminuye en esta fase. Cuando se supera el mínimo valor, el error suele aumentar luego la red se configura en el mínimo valor de la fase de validación.

Existe un último grupo que no interviene en la actualización de pesos y bias, es el llamado grupo de test. Este grupo evalúa la red como si se estuviera en la fase de identificación.

Generalizando este concepto a la educación, de un número finito de horas lectivas, se dispone de unas tantas para enseñar el temario, otras para que el alumno lo estudie y finalmente las dedicadas el examen.

Fin de entrenamiento

Como se ha detallado en la explicación de algunos algoritmos de entrenamiento, el objetivo de éste es recorrer la función de error buscando su mínimo absoluto. Esta tarea puede tardar enormemente y con alto coste de memoria. Por lo cual se establecen ciertos criterios para dar por bueno el entrenamiento aunque no se haya llegado al valor mínimo de la función.

- Gradiente: Mínimo valor del gradiente de la función de error.
- *Checks* de Validación: representa el número de iteraciones en la fase de validación que falla al decrementar el error.
- Error: Mínimo error cuadrático medio.
- Iteraciones: Máximo de iteraciones permitido.

Superado cualquiera de estos límites, el entrenamiento se da por finalizado.

3.3 Conclusiones

En este capítulo se ha abordado la problemática de la transcripción rítmica desde un punto de vista teórico. Partiendo de una señal de audio correspondiente a elementos percusivos, se deberá: adaptar la señal en un pre procesado, segmentar los eventos de interés, extraer las características (y posterior selección de éstas) para finalmente ser clasificadas y transcritas al lenguaje MIDI.

Se han introducido las bases de las redes neuronales artificiales como herramienta de clasificación. Detallándose el funcionamiento general que parte del elemento básico que componen cualquier red, la neurona.

Basándose en el comportamiento de una neurona biológica, se han descrito los procesos que se dan en ésta, para pasar a explicar la simplificación realizada en las neuronas artificiales. Los elementos clave de una neurona informática son la ponderación de cada entrada (peso), el umbral de variación (bias), y la función de activación o transferencia, la cual decide qué valor va a ofrecer la neurona a la salida. En cuanto al último elemento, existen numerosas funciones dependiendo del objetivo que se persiga, habiéndose descrito las más comunes.

Este elementos será la unidad básica para la construcción de las redes neuronales. La forma usual de conectar estas neuronas es formando capas. Las capas están definidas por el tipo de información que tienen a su entrada. De esta forma se establece un orden jerárquico tipo cascada en el que la salida de una capa es la entrada de la subsiguiente etc.

Existen infinitas topografías fruto de las posibles interconexiones de las capas neuronales, siendo éstas clasificadas en dos familias: estáticas y dinámicas o recurrentes. Las segundas poseen lazos de realimentación que conectan la salida de una capa con la entrada de capas anteriores; esto implica que el flujo de información no sea unidireccional (de la entrada a la salida de la red) sino que vaya fluyendo a través de las capas. La propiedad que estas realimentaciones introducen es la capacidad de tener memoria, ya que idealmente, la información estaría circulando por la red infinitamente. Esto es especialmente útil en el reconocimiento de secuencias, como podrían ser en el escenario en el que se trabaja: una secuencia de audio.

Este cerebro vacío creado, no tiene ninguna capacidad por sí solo si no es entrenado. Existen varias formas de enseñar a las redes neuronales, las cuales definirán los resultados de la red. El método supervisado consiste en entrenar a la red diciéndole si lo está haciendo mal o bien, y cuando ya es una experta en la tarea, podrá realizarla sin supervisión. Mientras que si se opta por el entrenamiento no supervisado, es la red la que deberá estimar si su comportamiento cumple el objetivo.

Los algoritmos de entrenamiento son los designados para enseñar a la red. Existen numerosos de ellos, describiéndose algunos de los más usados. Los cuales tienen en cuenta distintos criterios, siendo el más usado el de la minimización del error cuadrático medio *(mse)*.

PARTE II

Desarrollo del proyecto

Capítulo 4

Sistema de transcripción de voz a MIDI

En el proceso de la transcripción melódica, se busca transcribir al lenguaje MIDI, la interpretación cantada o tarareada lo más parecidamente posible.

Este objetivo se consigue siguiendo una serie de procesos que se detallan en la Figura 13. Primero se realizará una adaptación de la señal, para ser segmentada en los eventos que tengan interés para la transcripción. De éstos se obtendrán las características necesarias para conformar los mensajes MIDI en la última fase de codificación.

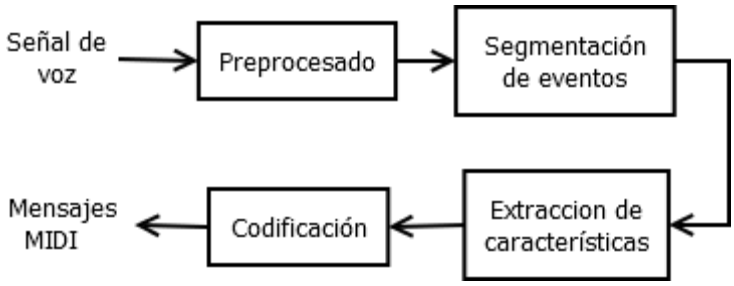

Figura 13: Etapas de la transcripción melódica.

4.1 Pre procesado

Esta fase busca adaptar la señal de entrada para facilitar el procesado de las siguientes etapas, más específicamente de la extracción de características. Es posible que la grabación de la señal de entrada se haya realizado en un ambiente poco silencioso o con un micrófono de baja calidad, o existan interferencias debidas a la red eléctrica, por lo que ésta tendrá ruido añadido. Además de este ruido que podría considerarse ambiental está el debido a la respiración del cantante cuando se encuentra cerca del micrófono.

Como forma de atenuar estos problemas se filtra la señal de entrada paso alto, teniendo en cuenta el límite mínimo de la voz cantada (80 Hz) se configura la frecuencia de corte del filtro a 70 Hz.

La función de autocorrelación en la que se basa el algoritmo de detección de tono contiene información de las resonancias propias del aparato fonador del cantante. Ocasionalmente, este filtrado acústico puede amplificar ciertas frecuencias lo sufriente como para que sean del mismo orden que los máximos debidos a la frecuencia fundamental. Estas componentes indeseables son de mayor frecuencia que F0, por lo que la influencia de éstas puede ser paliada con un filtrado paso bajo. Otro motivo por el que el filtrado paso bajo mejora el desempeño del algoritmo es la cualidad de aumentar la periodicidad de la señal. Con estas consideraciones se realiza un filtrado paso bajo con un filtro digital de orden 20, con frecuencia de corte 1000 Hz.

El tratamiento en esta fase, como ya se ha comentado depende enormemente de las necesidades de las siguientes etapas. En este caso, el algoritmo elegido para la obtención de tono que se explica en adelante, es robusto frente al ruido, por lo que basta con el sencillo filtrado explicado con anterioridad.

4.2 Segmentación de notas

Esta etapa es la encargada de detectar los inicios y finales de las notas producidas en la señal de audio. Siendo de vital importancia, ya que de nada sirve un buen sistema de cálculo de la frecuencia fundamental si no se detecta el evento.

Como se comento en la revisión de contenidos, existen varias formas de realizar ésta tarea entre ellas asignar cada evento a la posición de una rejilla temporal, o localizar los inicios (*onset*) de cada evento en función de sus características temporales y espectrales. Se va a tomar la segunda, dada la dificultad de establecer una rejilla si los datos se van analizando y reproduciendo en tiempo real.

En esta fase, se procede de diferente forma si el algoritmo trabaja en tiempo real, o si por el contrario, se desea que los eventos se graben en un fichero. El motivo es que en el segundo caso, se parte de una señal completa grabada y es posible operar sobre la envolvente, algo imposible cuando se dispone de una ventana de milisegundos.

Para ambas situaciones, el programa realiza una medición de ruido de fondo al iniciarse, con objeto de establecer un umbral real de ruido.

4.2.1 Tiempo real

El algoritmo de transcripción de voz en tiempo real va tomando tramos de señal de la entrada de micrófono de la tarjeta de sonido al mismo tiempo que la señal es generada por el usuario.

Cada ventana es procesada, antes de ser adquirida la siguiente. Por lo que es necesario tomar un tamaño de ventana que, por una parte sea lo más pequeño posible para que los datos lleguen rápidamente al sintetizador minimizando el tiempo de latencia, y por otra parte lo suficientemente grande como para que el procesado sea más lento que la adquisición de datos, y estos no se almacenen en el buffer.

Una condición a tener en cuenta en el tamaño de la ventana se deduce del método de detección de tono elegido basado en la autocorrelación (detallado más adelante). Teniendo en cuenta que la parte positiva de la autocorrelación tiene el mismo número de muestras que el tramo, y que la frecuencia fundamental se extrae del numero de muestras que se retardan hasta el máximo, si se toman ventanas de corta duración, se imposibilitara la detección de las frecuencias graves.

Considerando el mínimo tono del rango de voz en 78 Hz, y la frecuencia de muestreo 8000Hz, el tamaño mínimo de la ventana n viene dado por el siguiente desarrollo

$$\frac{1}{n\,T_s} = 78 \Rightarrow \frac{1}{n\frac{1}{f_s}} = 78 \Longrightarrow \quad n = \frac{f_s}{78} = 103\ muestras$$

Teniendo este límite en cuenta, y para evitar los molestos errores de octava, se establece un tamaño de ventana de 140 muestras, con una frecuencia de muestreo de 8000 Hz. Esto ofrece un rango de detección de pitch de 57 a 4000 Hz, ya que limite de máxima frecuencia de detección seria la situación de localizar el límite en la segunda muestra de la autocorrelación.

El problema del algoritmo en tiempo real, es que éste no puede hacer uso de la señal completa, observando las envolventes o características globales, ya que la señal no ha sido generada en su totalidad. Esto dificulta el segmentado de eventos, ya que se deben detectar los inicios y finales de las notas teniendo únicamente la información de la ventana presente (y de los datos ya grabados).

Con esto se entiende que lo primero que se debe hacer tras obtener una ventana y filtrar la banda de interés, es decidir a qué estado corresponde ese tramo de señal.

La función *det_estado* es una máquina de estados que se encarga de realizar esta tarea. Se debe distinguir entre el ″inicio″ de la nota (mensaje *note on*), el "fin" (mensaje *note off*) y las modulaciones en amplitud y frecuencia (mensajes *pitch bend y aftertouch*) que puedan haber entre los estados de ″inicio″ y "fin″ a la que se llamará estado "sostenido**".**

Estos estados están relacionados entre sí cumpliendo cierto orden, es decir, no se puede llegar al estado "sostenido" si no se parte de un estado de "inicio", o finalizar la nota. Esto se ilustra en la Figura 14.

El detector de estado hace uso de tres criterios para decidir el estado de la nota. Un umbral de energía, ya sea calculado en el inicio del programa o modificado por el usuario por medio de la interfaz gráfica.

La tasa de cruces por cero, la cual es alta para el ruido y baja para sonidos sonoros.

Transiciones desde "Inicio"

Se pasara a estado "sostenido" si la tasa de cruces por cero es menor al umbral sonoro/sordo y la energía actual supera en un 20% el umbral de ruido.

Si alguno de los anteriores criterios no se cumple, se pasara a "fin".

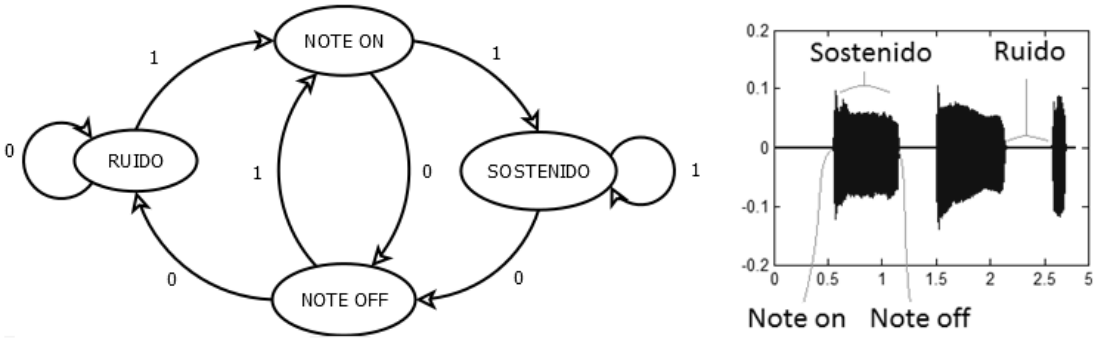

Figura 14: Izquierda: diagrama de estado del detector, en el cual los '1s' en las transiciones representa que se cumple el criterio, y '0' al contrario. Derecha: ejemplo de los estados en un tramo de voz

Transiciones desde "Sostenido"

Se mantendrá en estado "sostenido" mientras se siga manteniendo más de un 20% de la energía máxima almacenada y siempre que se trate de un tramo sonoro como indica la tasa de cruces por cero.

En caso contrario se pasara al estado de "fin".

Transiciones desde "Fin"

Se pasara a estado de "inicio" si los criterios de sonido sonoro y umbral de energía se cumplen; además, se toma el valor máximo de la derivada, cuyo valor será alto para inicios de nota.

Si no se cumplen los 3 criterios anteriores se pasara a estado de "ruido".

Transiciones desde "Ruido"

Si se cumplen los 3 criterios explicados en el párrafo anterior, se pasara a estado de "inicio".

Si no se cumplen, el detector de estado seguirá dando como resultado estado de "ruido".

En este caso de tiempo real, la extracción de características se produce cuando se llega al estado de fin de evento y se ha almacenado en memoria el tramo completo de la nota (ver diagrama de flujo en Anexo I Figura 56 y Figura 57).

4.2.2 Modo *Offline*

Al trabajar con la señal al completo (habiendo sido importada o grabada por el propio sistema), y poder realizar cálculos sobre la envolvente, se permite ofrecer una mayor calidad de segmentado.

El inicio de cada evento se manifiesta como un incremento de energía de la señal, esto conlleva un aumento en la envolvente de la forma de onda. Evaluando la derivada de la envolvente del tramo bajo estudio, se observaran máximos locales en las pendientes positivas (inicios de nota) y mínimos locales al decaer su energía (fin de nota). Estos máximos y mínimos formarán la base del sistema de detección de eventos.

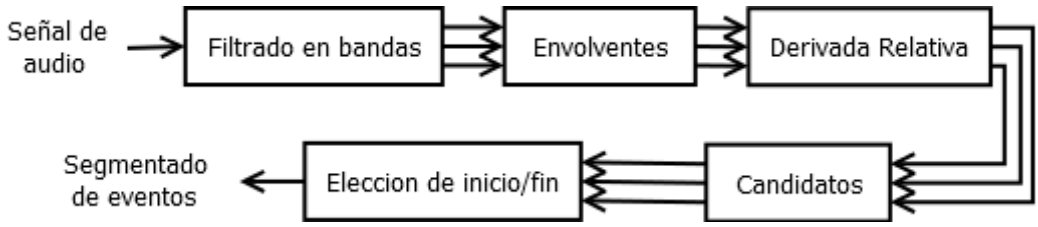

Figura 15: Procesos para obtener la segmentación de notas en el procesado off-line

Klapuri [7] señala la importancia de filtrar la señal de entrada en distintas bandas, para así aumentar la eficiencia del sistema.

Se parte pues de una señal, la cual es filtrada en 6 bandas, un filtro paso bajo hasta 127Hz, 4 filtros baso banda con frecuencias de corte 127 y 254 Hz, 254 y 508 Hz, 508 y 1016 Hz, y 1016 y 2032 Hz y un último filtrado paso alto a partir de 2032Hz. El ultimo filtro paso alto que propone [7], se obvia dado que el tono fundamental de la voz tiene su registro máximo alrededor de 1000 Hz

La derivada de la envolvente, es útil para reflejar la sonoridad de la nota [29], pero tiene el inconveniente de ser sensible a que modulaciones en el estado estacionario produzcan fuertes variaciones de la misma (ver Figura 16), incrementando así el número de detecciones erróneas.

Con objeto de subsanar este error, Klapuri propone el uso de la derivada relativa a la envolvente ecuación (14)

$$D_r(t) = \frac{\frac{d}{dt}E(t)}{E(t)} \tag{14}$$

donde E representa la envolvente de amplitud y t el instante de estudio.

Dado que las indeseables modulaciones de amplitud se encuentran en el sostenimiento de la nota, y por tanto el valor de la envolvente es alto, los picos en la derivada quedaran minimizados frente a los incrementos de energía que se produzcan en los inicios y fin de nota.

Como se observa en la Figura 16, los picos correspondientes al inicio del evento se han visto amplificados en el derivada relativa, y los debidos a molestas modulaciones casi extinguidos.

Tras el cálculo de la derivada relativa en cada banda, se establecen dos umbrales, uno positivo para los inicios de evento y otro negativo (en la derivada) para los finales. Los candidatos de cada banda son combinados para obtener la segmentación final.

Al estar basado en la derivada relativa, es posible que incrementos espurios de ruido produzcan incrementos en la derivada, y al encontrarse éste a muy bajo nivel, su derivada relativa se vea aumentada lo suficiente para superar el umbral de inicio. Como forma de eliminar este error se introduce el criterio de eliminar todos los eventos que tengan una duración menor de 60ms.

Cabe destacar que la obtención del pitch (que se pasa a detallar a continuación) se realiza de la misma forma que en el procesado en tiempo real. Pero la ventaja de hacer uso de este modo es que el usuario puede variar los parámetros como umbrales de energía, velocidad etc., posteriormente a la grabación y realizar el cálculo de la segmentación tantas veces como desee. Observando los resultados por medio de la interfaz grafica.

4.3 Extracción de características

En esta etapa se deben obtener las diferentes características que se necesitan para codificar el mensaje. Estas son:

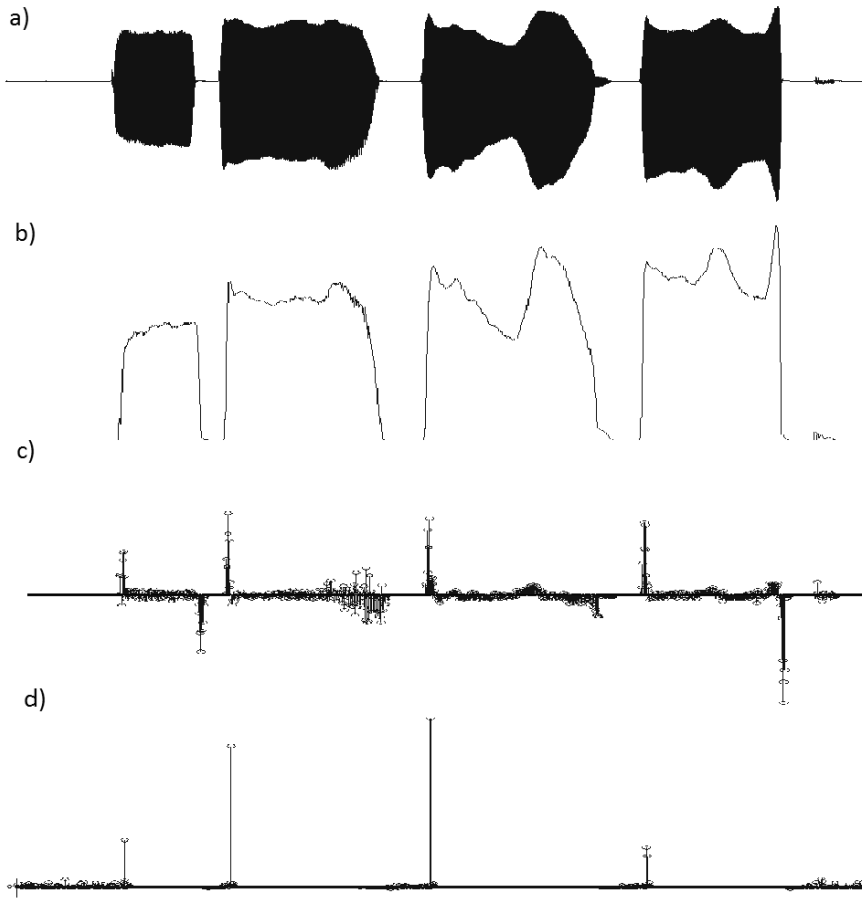

Figura 16: Proceso para la selección de los inicios (*onset*) de una banda de una señal de voz. La envolvente (b) es obtenida de la señal filtrada (a). De esta se calcula la derivada (c) donde se observan valores en las modulaciones que puedan interferir. La derivada relativa a la envolvente (d) muestra unos máximos más pronunciados, pudiéndose fácilmente umbral izar para la detección de los inicios.

4.3.1 Velocidad (*Velocity*)

Representa el volumen de la nota, su sonoridad. El rango de valores que este parámetro puede adoptar oscila entre 0 y 127, siendo el valor cero correspondiente a un silencio.

El cálculo de la sonoridad de una nota no se puede realizar simplemente observando la energía que ésta contiene, según [7] el parámetro que mejor refleja la sonoridad de una nota está en el ataque, en la derivada de primer orden de la envolvente. Klapuri sostiene que una nota con mayor ataque y menor energía global produce una sensación de mayor volumen sonoro que un alto sostenimiento con bajo ataque.

El modo de obtener la velocidad se basa en una transformación del valor de la derivada de primer orden del ataque de la nota, a una escala de 128 valores.

La proporción de derivada mínima a derivada máxima (pendiente de la función de transferencia), puede ser modificada por la interfaz grafica, con objeto de que el usuario pueda aumentar o disminuir la dinámica a su gusto.

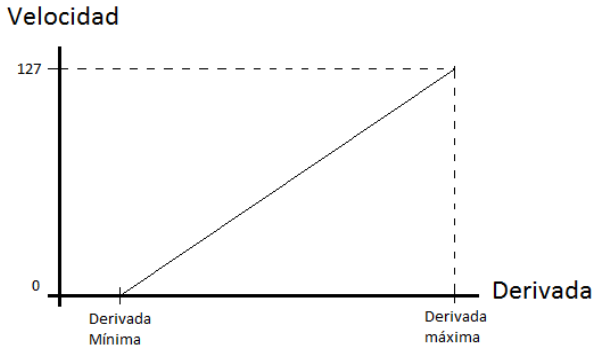

Figura 17: Función de transferencia en la conversión del valor de la derivada al valor de velocidad MIDI.

4.3.2 Tono

El objetivo que se persigue es el de encontrar el tono fundamenta de una nota producida por la voz haciendo uso de un detector automático.

Dado que se busca que la aplicación trabaje en tiempo real, el algoritmo debe ser lo más rápido posible.

Por lo general, los algoritmos que operan en el dominio transformado (cepstrum o HPS), tienen un coste computacional mayor y por lo tanto, un tiempo mayor de respuesta. Esto hace centrar la atención en los métodos de detección en el dominio del tiempo, más específicamente en la autocorrelación. Además que éstos son los que recientemente han obtenido mejores resultados en aplicaciones de detección de tono por tarareo ([8], [9]).

Dado un tramo de señal de voz del que se desea extraer el tono fundamental. Se va a obtener la autocorrelación cruzada multiplicando dicha señal por réplicas desplazadas como se ha explicado en la descripción general.

De esta función de autocorrelación se buscará el máximo, cuya diferencia con el desplazamiento cero se corresponderá con el periodo fundamental. Basta con realizar la inversa de dicho valor para obtener la frecuencia fundamental (F0).

Debido a la naturaleza propia de la autocorrelación, ésta presenta menor precisión conforme el tono se va tomando valores de más alta frecuencia. Basta con observar la precisión entre dos muestras adyacentes de la función de autocorrelación, en diferentes puntos de la misma. Se pasa a realizar un ejemplo de cálculo sobre el tramo de señal de voz de la **Figura 18**.

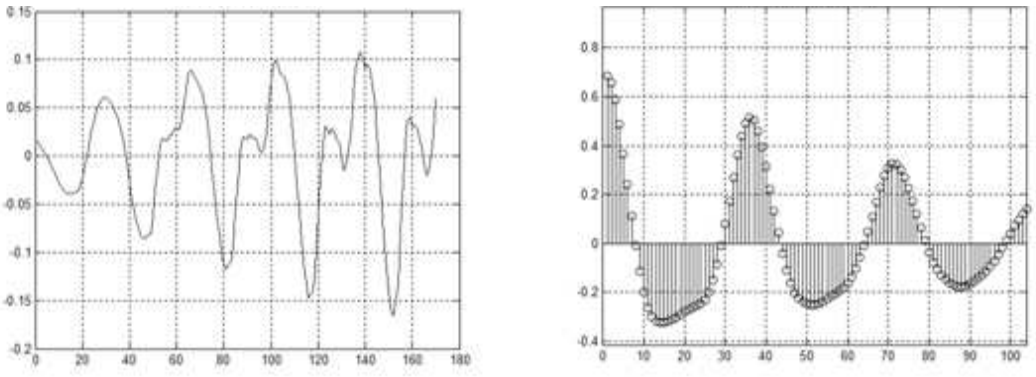

Figura 18: Tramo de voz de F0 = 227.5Hz (izquierda) y parte positiva de la función de autocorrelación (derecha).

- Mayor frecuencia, muestras 36-37.

$$precision\ (Hz) = f_{muestra\ 36} - f_{muestra\ 37} = \frac{1}{T_0 * 36} - \frac{1}{T_0 * 37} \overset{f_s=8000Hz}{\cong} 222.22 - 216.22 = \mathbf{6Hz}$$

- Menor frecuencia, muestras 71-72.

$$precision\ (Hz) = f_{muestra\ 71} - f_{muestra\ 72} = \frac{1}{T_0 * 71} - \frac{1}{T_0 * 72} \overset{f_s=8000Hz}{\cong} 112.67 - 111.11 = \mathbf{1.55Hz}$$

Esta peculiaridad es útil dado el rango de frecuencias que ocupa la voz.

Sobre la precisión

Ahora bien, como se ha comentado en el apartado anterior, el desplazamiento correspondiente al periodo fundamental es un número discreto de muestras. En la situación en la que el usuario interprete un tono cuya frecuencia va aumentando, la respuesta del sistema sería escalonada y no continua (algo inherente a los sistemas digitales). Esta es una situación que no se puede corregir, pero puede paliarse minimizando los "saltos" de la respuesta. Además de este inconveniente está la posibilidad de que se asigne un valor de nota erróneo, por falta de precisión de algoritmo.

Esta precisión en la deteccion de pitch depende directamente del número de muestras que representen un mismo periodo, y por lo tanto de la frecuencia de muestreo.

Una posibilidad podria ser aumentar la frecuencia de muestreo, esto implicaría capturar un mayor número de muestras en la entrada, por lo que se aumentaría proporcionalmente el tiempo de respuesta del sistema. Otra posibilidad es interpolar el tramo de entrada, de esta forma no se pierde tiempo en obtener excesivas muestras en la captacion.

Pero como se propone en [10] ésto supone un aumento del coste computacional al calcular la autocorrelacion, por lo que se propone interpolar cúbicamente la función de autocorrelación, en el rango de valores $[\tau - 1, \tau + 1]$, siendo τ la posición del máximo.

El diagrama de bloques correspondiente al algoritmo se muestra en la Figura 19:

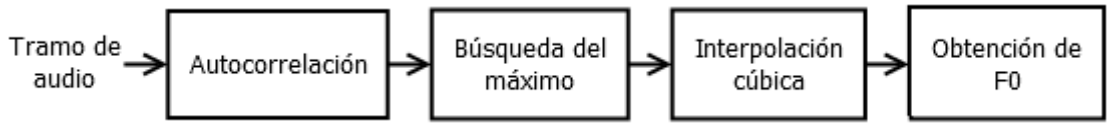

Figura 19: Diagrama de bloques de los procesos para el cálculo de la frecuencia fundamental F0.

Los molestos errores de octava, propios de este sistema de deteccion de tono se palian con un filtrado de mediana a lo largo de la nota. Esto se traduce en una reduccion de la precision temporal en la variacion del tono, pero se eliminan los picos espurios correspondientes a los errores de octava y no se varía el valor del tono general obtenido.

4.4 Codificación

Extraídas las características de interés de la señal de voz como son

La codificación es la conformación de los mensajes MIDI correspondientes a las características obtenidas en las anteriores etapas de procesado.

La frecuencia de cada nota deberá ser codificada en un valor de nota MIDI dado por la ecuación:

$$nº\ nota\ MIDI = 69 + 12\log_2\frac{F_0}{440\ Hz} \quad (15)$$

donde 69 y 440 Hz corresponden al número de nota MIDI y frecuencia fundamental del LA de la 4ª octava respectivamente.

La obtención de la velocidad o volumen de la nota, se ha detallado en la extracción de características.

En el caso en el que se haya habilitado la rueda de modulación de tono (*pitch bend*), la variación del tono durante la nota será codificada en el rango de valores [-8192, 8192], siendo el *0* equivalente a ninguna modulación, valores negativos a un decremento de tono y viceversa.

La variación de tono que produce la rueda de modulación varía dependiendo del sintetizador que se esté utilizando, es decir, una variación de 0 a 1000 implica la variación de medio tono para un sintetizador, y de un tono para otro. Esto conlleva el no poder realizar una transcripción única de la variación de la voz a estos mensajes. La solución adoptada en este proyecto es el de habilitar la posibilidad de que el usuario varíe la velocidad a la que se modifica el tono por medio de la interfaz gráfica y éste la ajuste al sintetizador que esté usando.

En el caso de trabajar en tiempo real, los eventos son mandados al sintetizador según son obtenidos. En la transcripción offline, éstos deben ser cuantificados y ordenados temporalmente. El formato de fichero elegido por su simplicidad es el de MIDI texto (ver ejemplo en Anexo II). En éste, tras las cabeceras de fichero y track, se irán grabando los eventos con sus correspondientes características obtenidas.

Finalmente, este fichero de texto deberá ser ensamblado a un fichero con formato MIDI (.mid) mediante cualquier ensamblador del mercado, en este caso se recomienda el software MIDI DisAssembler. El fichero ya en el formato estandarizado puede ser utilizado en cualquier secuenciador.

Como ya se ha comentado, es común en las interpretaciones de voz cantada sin acompañamiento, que el interprete varíe la afinación de referencia a lo largo de la grabación. Esto implica que notas inicialmente afinadas con respecto al LA de 440Hz (por ejemplo), acaben referenciándose a un supuesto LA de 430 Hz. Esto al oído humano no supone tanto, pero al convertir los tonos a notas MIDI, el resultado es una interpretación completamente diferente a la deseada.

En este proyecto se supone que el cantante no va a seguir ninguna afinación constante, por lo que en vez de evaluar el tono que éste está interpretando, se observará el intervalo con respecto a la anterior nota. De esta forma se palia la falta de constancia en la referencia.

De esta forma, el error máximo en un intervalo será de medio tono, el cual se arrastrará hasta el final de la grabación. Esto es fácil de cambiar en cualquier secuenciador, seleccionando el tramo "desafinado" y transponiéndolo un semitono.

Esta suposición en la afinación, se implementa únicamente en el algoritmo *offline*. Esto es debido a que el cambio de referencia implica mandar otras notas al sintetizador, esto, al ser escuchado en tiempo real produce desorientación y es contraproducente. Por este motivo, en la transcripción en tiempo real, los datos MIDI son los correspondientes a la nota más cercana al tono fundamental de la voz.

4.5 Conclusiones

En este capítulo se ha abordado la transcripción de voz a MIDI. Partiendo de un esquema global del procesado se ha descrito cada una de las fases necesarias para conseguir este objetivo.

Se ha observado como el objetivo del pre procesado depende de las necesidades de las siguientes etapas, proponiéndose en este caso un filtrado de la banda de interés únicamente.

El sistema implementado abarca tanto el tiempo real como el procesado *offline*. Siendo la principal diferencia en el procesado la correspondiente a la etapa de segmentado de eventos.

En primer lugar, se ha definido la captación de datos en la aplicación de tiempo real, evaluando la problemática del tamaño de ventana. Para pasar a explicar el segmentado de eventos basado en la clasificación de cada ventana obtenida con una maquina de estados.

En el caso del segmentado *offline,* se introduce un robusto método de detección de eventos basado en la derivada relativa a la envolvente de la amplitud.

El tono o frecuencia fundamental es obtenido mediante el algoritmo implementado que se basa en el método de la autocorrelación, analizándose los problemas de precisión y detección errónea que éste presenta. Se proponen ciertas modificaciones del algoritmo clásico de autocorrelación con el objetivo de aumentar la precisión y eficiencia del mismo.

Además del tono de la nota de voz, se debe obtener la velocity (volumen) de la nota. Calculándose dicha sonoridad en base a la derivada de primer orden del ataque.

Habiéndose descrito la forma en la que se calculan los diferentes parámetros, éstos son convertidos a mensajes MIDI, introduciéndose la problemática de la inclusión de la rueda de modulación (*pitch bend*), la cual depende fuertemente del sintetizador que se utiliza.

También se discute la elección de la referencia de afinación en el algoritmo *offline*, tomándose la decisión de partir de que el usuario no va a mantener ningún tipo de referencia, codificándose los intervalos interpretados y no la frecuencia de la nota. En caso de el algoritmo en tiempo real, no es posible realizar este ajuste de afinación tomándose la referencia absoluta del LA temperado.

Capítulo 5

Sistema de transcripción de impactos a MIDI

5.1 Visión General del sistema

En este capítulo se aborda la transcripción rítmica desde el punto de vista práctico. Los ritmos que el usuario va a interpretar serán producidos por los golpes de sus propias manos, un lápiz o una cuchara como baquetas improvisadas, sobre combinaciones de objetos como cajas, botes, vasos, o la mesa, por ejemplo.

La herramienta que va a clasificar entre los distintos golpes va a ser una red neuronal artificial, ya introducida en la revisión de conocimientos. Tras conocer la clase a la que pertenece dicho gesto, se conformará el mensaje MIDI.

Dado que se busca que la aplicación trabaje tanto en tiempo real como en no real, se van a diseñar dos redes, cada una de ellas orientada a conseguir los mejores resultados en ambas situaciones. Se trabajará sobre una red dinámica de rápida respuesta, y una estática con mayor eficiencia pero mayor tiempo de procesado.

Antes de comenzar con el diseño de las redes neuronales, se pasa a describir el procedimiento genérico que se ha implementado, para la grabación del set de entrenamiento, y el posterior uso de la red en la fase de identificación.

Configuración previa

Para el correcto funcionamiento del sistema, se debe disponer de un micrófono (la calidad no es imprescindible) enchufado a la entrada de micrófono del ordenador (ver Figura 20). El dispositivo de captación debe estar orientado lo más posible al centro de la escena de elementos que se van a golpear.

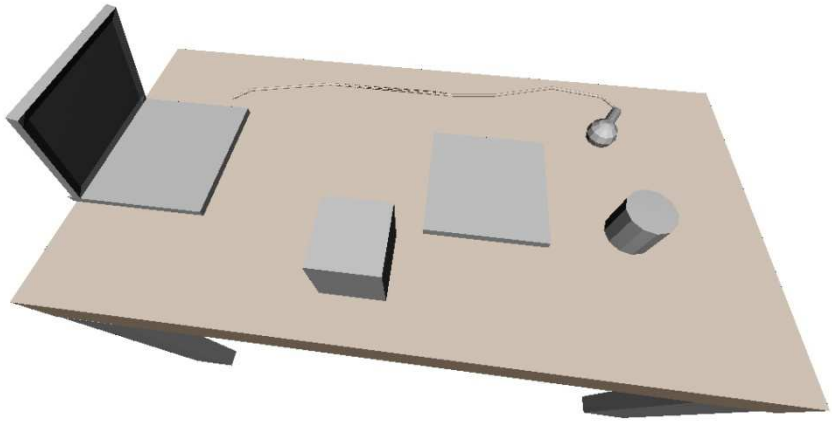

Figura 20: Ejemplo de la disposición de elementos, para la transcripción rítmica.

Es preferible que previamente se observen los niveles de señal que ofrece el micrófono, para que éstos no saturen, ni su energía sea inexistente. De todas formas, los golpes sobre las superficies no van a ofrecer un alto nivel de señal, por lo que generalmente no darán problema a la entrada de la tarjeta de sonido del ordenador.

Una vez dispuestos los elementos, se inicia el programa.

5.2 Fases de la transcripción rítmica

La transcripción se va a realizar con el objetivo de diferenciar 4 clases, que el usuario podrá asignar vía MIDI a cualquier elemento percusivo de su elección.

El uso de redes neuronales artificiales para la discriminación de clases implica necesariamente una fase de entrenamiento. Los estadios que intervienen se detallan en la Figura 21.

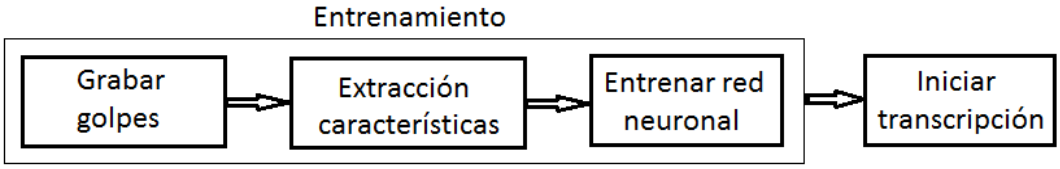

Figura 21: Etapas de la transcripción rítmica.

Como se ha comentado, se proponen dos tipos de redes neuronales para atacar el problema.
Esto implica la extracción de dos familias de características, una para la red dinámica y otra para la estática.

5.2.1 Grabar golpes

En esta fase se busca crear una base de datos que pueda ser utilizada en la configuración de los parámetros de entrenamiento de las redes neuronales.

Al accionarse la grabación de una clase, el sistema toma un número determinado de ventanas de ruido. La energía máxima registrada del tramo de ruido es guardada como umbral para su posterior uso.

Ya grabado el ruido, se informará al usuario que debe golpear cada superficie repetidas veces (un total de 40). Estos serán grabados y clusterizados, para extraer las características oportunas y así entrenar las redes que el usuario haya elegido para la discriminación de clases.

La toma de datos se realiza tomando ventanas de señal, cada ventana es analizada con una maquina de estados, la cual decidirá a qué se corresponde; inicio o fin de evento, o por el contrario es ruido carente de información. (ver diagrama de flujo en Anexo I Figura 54).

Detección de estado

Las señales de entrada van a ser de origen percusivo, por lo que tendrán un fuerte ataque, un inexistente sostenimiento y un decaimiento también pronunciado. Estos cambios súbitos de energía conllevan que la derivada sea un poderoso medio de evaluación. Valores altos de derivada marcaran el inicio del golpe.

Además de la derivada, el sistema almacena el valor máximo de energía que se da en el golpe, para establecer el umbral de decaimiento con respecto a éste (para el fin de nota). El detector también tiene en cuenta la energía de la ventana y el estado precedente. Este dato es de gran relevancia, ya que la sucesión de estados tiene un orden concreto; no se puede llegar a un fin de nota sin que se haya dado previamente un inicio, por ejemplo.

El diagrama de estados del detector que se ha descrito se muestra en la Figura 22.

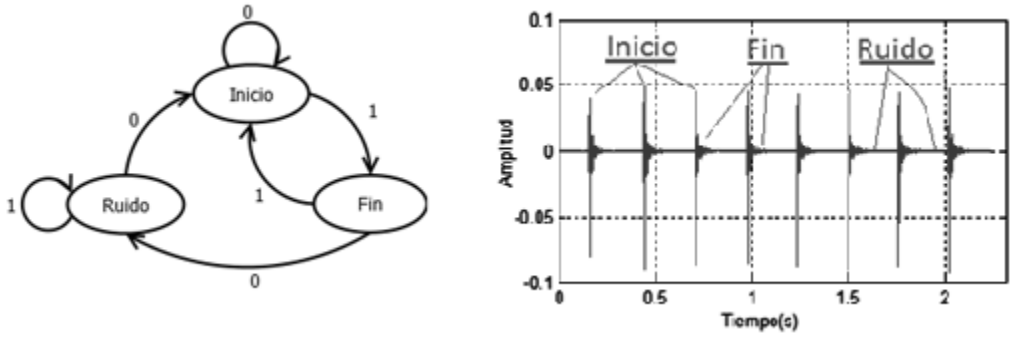

Figura 22: Diagrama de estados (izquierda).Señal captada de choques de lapicero contra mesa (derecha), detallándose los estados.

Se pasa a describir las condiciones de transición entre estados.

Ya que se realiza una medición de ruido previa al golpeo, siempre se partirá del estado *ruido.*

Transiciones desde RUIDO

El detector de estado se mantendrá en ruido hasta que la energía de la ventana supere el umbral de energía calculado en la medición de ruido, y la derivada del tramo supere un umbral de derivada establecido.

Si las anteriores condiciones se cumplen, se pasara a estado de *inicio*, en caso contrario, se mantendrá en *ruido.*

Transiciones desde INICIO

Puede darse el caso de golpes sobre superficies que radien suficiente tiempo como para ocupar dos o más ventanas, por lo que se mantiene el estado de *inicio* mientras la energía no caiga por debajo del 20% del valor máximo del tramo grabado desde la primera ventana

etiquetada como *inicio*. De esta forma, se obtendrán varias ventanas correspondientes a un mismo golpe.

En el momento en el que no se supere este 20%, se pasa a estado de *fin*.

Transiciones desde FIN

Los criterios de umbral de energía y derivada que se siguen en la transición de *ruido* a *inicio* se mantienen proviniendo desde *fin*.

En el caso de que no se superen ambos umbrales, el detector mandará como estado *ruido*.

Tras conocer si nos encontramos en un inicio de nota, un fin o la ventana solo alberga ruido, se procede a procesar la ventana de diferente forma dependiendo del estado. (ver diagrama de flujo en Anexo I Figura 54).

Ventana en estado de INICIO

Cuando una ventana es catalogada como inicio, ésta se guarda en una variable auxiliar, y su energía máxima es almacenada. En el caso de que la siguiente ventana tomada también contenga señal útil, implicará que un mismo golpe ocupa dos ventanas y ambas serán unidas para conformar el golpe, la energía máxima es comparada y la más alta es guardada. Esto sucederá para todas las ventanas con estado de inicio consecutivas.

Ventana en estado de FIN

Si el detector informa de que la ventana actual corresponde al final de un golpe, y el estado anterior de señal era inicio, se procede a procesar el golpe.

La red dinámica o recurrente no requiere de ningún procesado, ya que tiene como entrada un tramo de la señal temporal. Por lo que, tras una normalización, se pasa a copiar el número correspondiente de muestras del ataque en la matriz que compondrá los datos de entrenamiento.

Para la red dinámica se obtendrán todas las características mencionadas mediante la función *get_features,* y éstas se guardarán en la variable que contendrá las características de cada golpe, para su posterior selección.

Ventana en estado de RUIDO

En el caso de ser ruido, se procederá a tomar otra ventana.

5.2.2 Entrenar la red neuronal

Una vez grabadas todas las entradas, y extraídas las características de cada golpe (ver características pagina 54). Se procede al entrenamiento de la red neuronal.

Ya que los algoritmos a utilizar son de naturaleza supervisada, es necesario conocer la clase a la que pertenece cada golpe. Esto es una tarea sencilla, ya que el usuario ha grabado los golpes correspondientes a cada clase, de forma claramente diferenciada.

La matriz de salidas deseadas (*targets*) se compone por 4 filas (correspondientes a las cuatro clases), y las columnas resultantes de la suma de todos los golpes. Cada columna representará un golpe dado, y el máximo valor de ésta será la clase a la que pertenece. Se pasa a detallar un ejemplo:

Teniendo que diferenciar entre cuatro clases y habiendo grabado dos golpes de cada una de ellas (en la práctica serán mas), se conforma la matriz de entradas y *targets* (Tabla 1).

El entrenamiento consistirá en aplicar a la entrada de la red neuronal una de las columnas de la matriz de entradas, y compararla con la salida deseada, el error se utilizara para variar los pesos y bias como se ha comentado en la revisión de conocimientos.

Cuando la red es utilizada en la fase de identificación, su salida será un vector columna con los 4 valores. a columna cuyo valor esté más próximo a 1 corresponderá a la clase elegida por la red.

Realmente, no todos los datos son usados para el entrenamiento, éstos son separados en los distintos grupos: entrenamiento, validación y test.

Los procesos de selección de las características y condiciones de entrenamiento son propias del diseño de cada red, por lo que se detallan en los capítulos posteriores (putos 5.4 y 5.5).

Tabla 1: Matriz de entradas para entrenamiento (arriba), y su correspondiente matriz de salidas deseadas (abajo).

	Clase 1		Clase 2		Clase 3		Clase 4	
	Golpe 1	Golpe 2	Golpe 1	Golpe 2	Golpe 1	Golpe 2	Golpe 1	Golpe 2
Características	2,735	1,73	0,807	0,572	2,246	2,222	2,279	2,339
	12,64	6,47	2,75	2,41	6,56	6,65	8,26	9,13
	129	207	14	204	29	31	256	208
	0,48	0,39	0,4	0,35	0,12	0,11	0,61	0,53
	6,22E-05	5,85E-05	5,25E-06	4,74E-06	5,61E-05	9,92E-05	2,63E-05	2,97E-05

	Salida deseada							
Clase 1	1	1	0	0	0	0	0	0
Clase 2	0	0	1	1	0	0	0	0
Clase 3	0	0	0	0	1	1	0	0
Clase 4	0	0	0	0	0	0	1	1

5.2.3 Iniciar transcripción

Una vez grabados los golpes, y habiéndose entrenado la red. Solo falta elegir la salida de datos (reproducción en tiempo real, o escritura en fichero) y comenzar la transcripción.

El algoritmo de transcripción rítmica guarda ciertas similitudes con la grabación y análisis de los golpes previo al entrenamiento (Anexo I Figura 54).

En este caso, el umbral de ruido no se graba al inicio cada vez que se presione el botón "INICIAR", sino que se toma el ultimo umbral de la grabación anterior suponiendo un ambiente acústico de escaso cambio. Esto evita que cada vez que el usuario quiera interpretar un ritmo deba esperar a la medición de ruido, y sobre todo, si éste desea grabar la interpretación tocando sobre una melodía, es importante que cuando se accione el botón de inicio comience la transcripción, para su posterior sincronización. (ver diagrama de flujo Anexo I Figura 55).

Al tomarse una ventana, se decide el estado en el que se está, de la misma forma explicada anteriormente. Nuevamente, dependiendo del estado el proceso es diferente.

Ventana en estado de INICIO

Se almacena la ventana actual en un array con objeto de conformar el golpe completo. También se almacena el valor máximo de energía, para utilizarse en la detección del final del evento.

Ventana en estado de FIN

Este estado significa que se ha pasado por el estado de inicio, y se tiene guardado en memoria el tramo de señal.

Lo más posible es que el inicio del golpe no coincida exactamente con el inicio de la ventana tomada, por lo que es necesario eliminar los tramos de ruido que puedan existir antes del inicio del golpe, y también el final del mismo. Ya que esto afectaría a la robustez de las características espectrales, y temporales. Sobre todo a la red dinámica cuya entrada es la señal temporal.

Teniendo ya preparada la señal sin ruido, se pasa a obtener las características dependiendo de la red que se haya elegido por medio de la interfaz grafica:

- Red estática: Se extraen las características que ya han sido marcadas por el algoritmo de selección (ver página 84) en la fase de entrenamiento.
- Red dinámica: se toma el tramo de ataque de interés.

La red neuronal ya entrenada decide a que clase se corresponde.

Una vez elegida la clase, se obtiene la velocity del golpe interpretado. Recordemos que en la fase de entrenamiento, se guarda el valor medio de energía de cada clase, por lo que la velocity se calculará en función de este valor.

Tener una referencia distinta para cada clase puede parecer algo carente de sentido. Pero cuando cada clase puede estar interpretada en una superficie diferente, la radiación sonora de estos elementos puede diferir enormemente, produciendo que un elemento percusivo suene más que otro sin que esto represente los deseos de la interpretación del usuario.
Esto se soluciona, como ya se ha comentado, tomando un valor de referencia diferente para cada clase, calculado en base al valor medio de la energía grabada en la fase de entrenamiento.

Finalmente y en función de la salida de datos elegida, se mandarán los datos MIDI a un fichero de texto, o bien al sintetizador. Además, si se ha elegido la escritura, el sistema cerrará el fichero al pulsar el botón detener.

5.3 Redes neuronales: Fundamentos de diseño

A la hora de adentrarse en el diseño de una red neuronal, uno de los problemas que se puede observar es la gran cantidad de combinaciones de parámetros que afectan al comportamiento de ésta.
El objetivo de realizar un estudio sobre los parámetros de diseño es doble ya que al intentar establecer un orden de importancia en la toma de decisiones, es necesario jugar con las distintas variaciones alumbrando el camino de las posibles redes neuronales que finalmente se vayan a usar.

A rasgos generales, el diseño de una red neuronal artificial consiste en la toma de las siguientes decisiones:

- Tipo de red: recurrente o estática
- Número de capas
- Número de neuronas por capa
- Funciones de transferencia de cada capa
- Elección de retardos (si la red es recurrente)
- Algoritmo de entrenamiento

Antes de tomar ninguna decisión es imperativo el conocer la relación de estos parámetros entre sí, para establecer el orden de importancia que se comentaba previamente con objeto de optimizar el diseño de la red.

Algunas de estas relaciones son evidentes, se debe decidir cuantas capas va a tener la red antes de establecer el número de neuronas por capa. Pero la forma en la que afecta el algoritmo de entrenamiento a las funciones de activación no es algo tan intuitivo.

Pero hay algo que sí que se puede saber de antemano: el número de elementos en las capas de entrada y salida. La capa de entrada tendrá tantas neuronas como muestras se vayan

a introducir en la red y el número de neuronas de la capa de salida corresponderá (en esta aplicación) con el total de clases a distinguir, 4.

Se va a realizar un estudio de la interrelación de las variables de diseño mencionadas, para terminar proponiendo un modelo general orientativo de diseño.

Dadas las diferencias entre una red dinámica y una estática, se van a estudiar por separado.

5.3.1 Red estática

Como ya se ha comentado, las redes estáticas son aquellas cuyo flujo de información tiene una única dirección, de la entrada a la salida. Sin bucles de realimentación.

Feed-Forward de una capa oculta

Se va a partir de la estructura más simple: una red de propagación hacia delante (*Feed-Forward*), de una única capa y manteniendo la función lineal en su capa de salida.

Los parámetros a evaluar serán:

- Funciones de activación: Tan-sigmoidea (Tansig), Log-sigmoidal (Logsig) y lineal (Purelin)
- Algoritmo de entrenamiento: Conjugate Gradient with Powell/Beale Restarts(CGB), Resilient Back propagation(RP), Polak-Ribieri Conjugate Gradient (CGP) y Levenberg-Marquardt (LM).
- Número de neuronas en capa oculta.

Para cada algoritmo se prueban las diferentes combinaciones con las anteriores variables. Ya que las redes neuronales ofrecen distintos resultados cada vez que son entrenadas, cada topología pasa por el proceso de aprendizaje 15 veces, para luego promediar los resultados de la actuación de la red, en este caso, se evaluara el error cuadrático medio (*mse*).

El *mse* mide la diferencia entre el resultado obtenido por la red y el deseado. Los resultados se muestran en la
Figura 23. Se puede observar en la gran variación de las curvas que la forma en la que se entrene la red neuronal es de vital importancia.

Como ya se había comentado en la fundamentación teórica, la función lineal en las capas ocultas carece de interés, como se puede observar es la que obtiene mayor error independientemente del algoritmo a testar.

Para cada algoritmo, existe una función que minimiza el error. **Luego la elección de la función de transferencia depende del algoritmo elegido**.

Si se evalúa la relación entre las funciones de transferencia (tansig y logsig) y el número de neuronas, se observa que para ambas funciones , el error se minimiza en un mismo número de neuronas. Luego se puede realizar la aproximación siguiente: la función de transferencia es aproximada ente independiente del número de neuronas para esta topología.

El número de neuronas que minimiza el error se encuentra en 20 (25 para CGB). Hay que tener en cuenta que se han tomado en saltos de 5 elementos, por lo que es probable que para cada algoritmo exista un número optimo de neuronas.

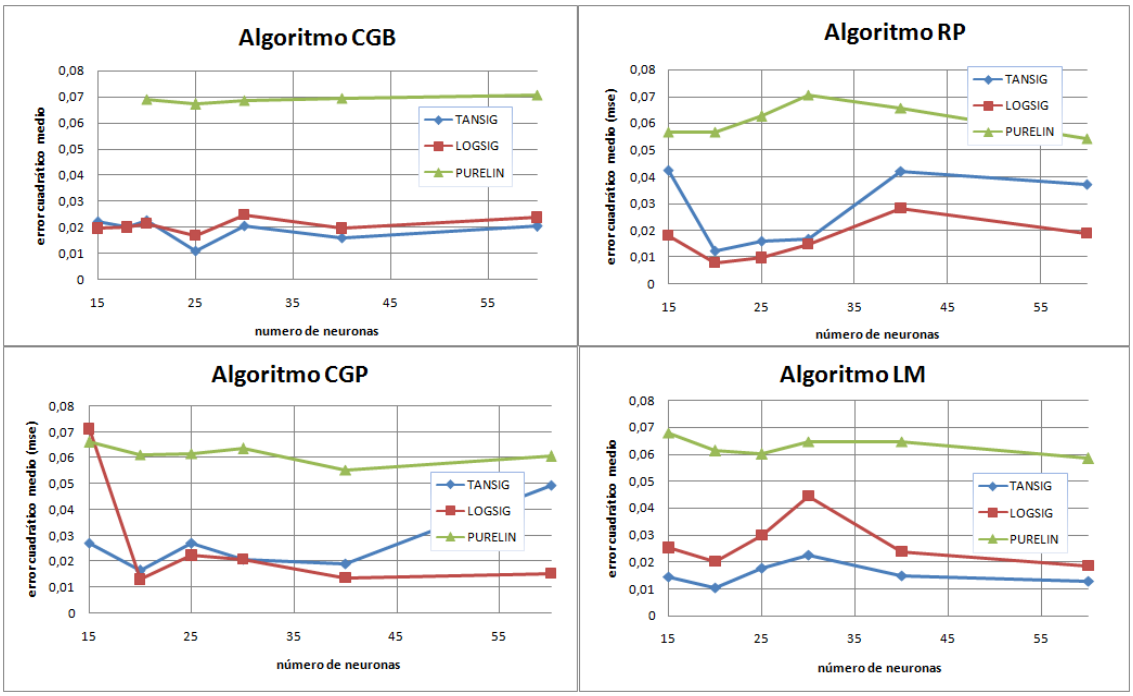

Figura 23: Diferentes combinaciones de algoritmo y funciones de activación en función del número de neuronas en la capa oculta.

Feed-Forward de dos capas ocultas

La inclusión de una segunda capa fuerza el estudio de la relación entre el número de neuronas en cada capa, y de la combinación entre funciones de activación.

Los resultados del estudio de dos redes de 15 y 25 neuronas en la primera capa se muestran en la Figura 25 .

Para los tres algoritmos analizados, la combinación de funciones de transferencia que minimiza el error varia con el número de neuronas en cada capa. Por ejemplo, para RP y 15 neuronas en la primera capa, es la combinación tansig-logsig la que tiene menor error , pero al aumentar a 25, es tansig-tansig la optima. Lo cual indica que se deberá elegir el número de capas y neuronas por capa previamente a la decisión de la combinación de funciones de transferencia.

Metodología de diseño: red estática

Con lo estudiado en las topologías de una y dos capas, se puede concluir que la toma de decisiones de diseño seguirá lo siguientes pasos:

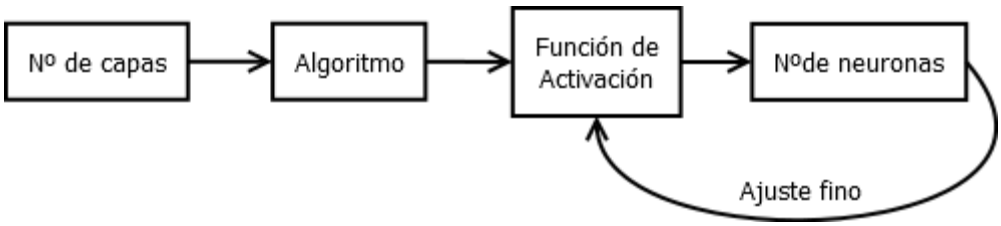

Figura 24: Diagrama de bloques orientativo de la toma de decisiones en el diseño de una red estática.

En primer lugar se deberá elegir el número de capas a tener en cuenta. A continuación, decidir qué algoritmo entrenara la red. Se procederá a elegir la/s funciones de activación y finalmente el número de neuronas que minimicen el error medio.

Se ha de decir que esto no es un camino lineal a recorrer para el diseño, sino una visión general y orientativa. El bucle de ajuste fino que se muestra en la figura representa la posibilidad (por no decir seguridad) de tener que modificar pasos anteriores en pro de optimizar la red.

5.3.2 Red dinámica

En contraposición a las redes estáticas, las redes recurrentes o dinámicas tienen lazos de realimentación. Por lo que el flujo de información va pasando de una capa a otra y realimentándose.

Por lo tanto, se podría considerar que el la información está en continuo movimiento por la red. Comparativamente, si una red estática representa un sistema FIR (*finite impulse response*) la red dinámica seria IIR (*infinite impulse response*) .

Red de Elman de una capa

Se procede a observar cómo se relacionan el algoritmo de entrenamiento con las funciones de activación y el retardo elegido. Se ha omitido la función lineal como posibilidad por los malos resultados experimentados.

Algoritmo LM

La inclusión de memoria en el sistema no parece mejorar los resultados, pero produce que sea necesario un mayor número de neuronas para minimizar el error. Cabe decir que existe un máximo de memoria disponible por Matlab, y conforme aumentamos la el número y valor de los retardos, este límite de memoria es sobrepasado, dando error, por ello no se ha superado el umbral de 50 neuronas para *n=1,2* y 40 para *n=1,2,3*. Como se observa, el límite es menor cuanto mayor es el retardo.

También hay que tener en cuenta que el tiempo dedicado al entrenamiento de una red de 40 neuronas y retardo 1,2,3 es de más de un minuto, tiempo excesivo de espera para el usuario.

La función de transferencia en la capa oculta tampoco parece afectar a la minimización del *mse*, dando *tansig* mejores resultados.

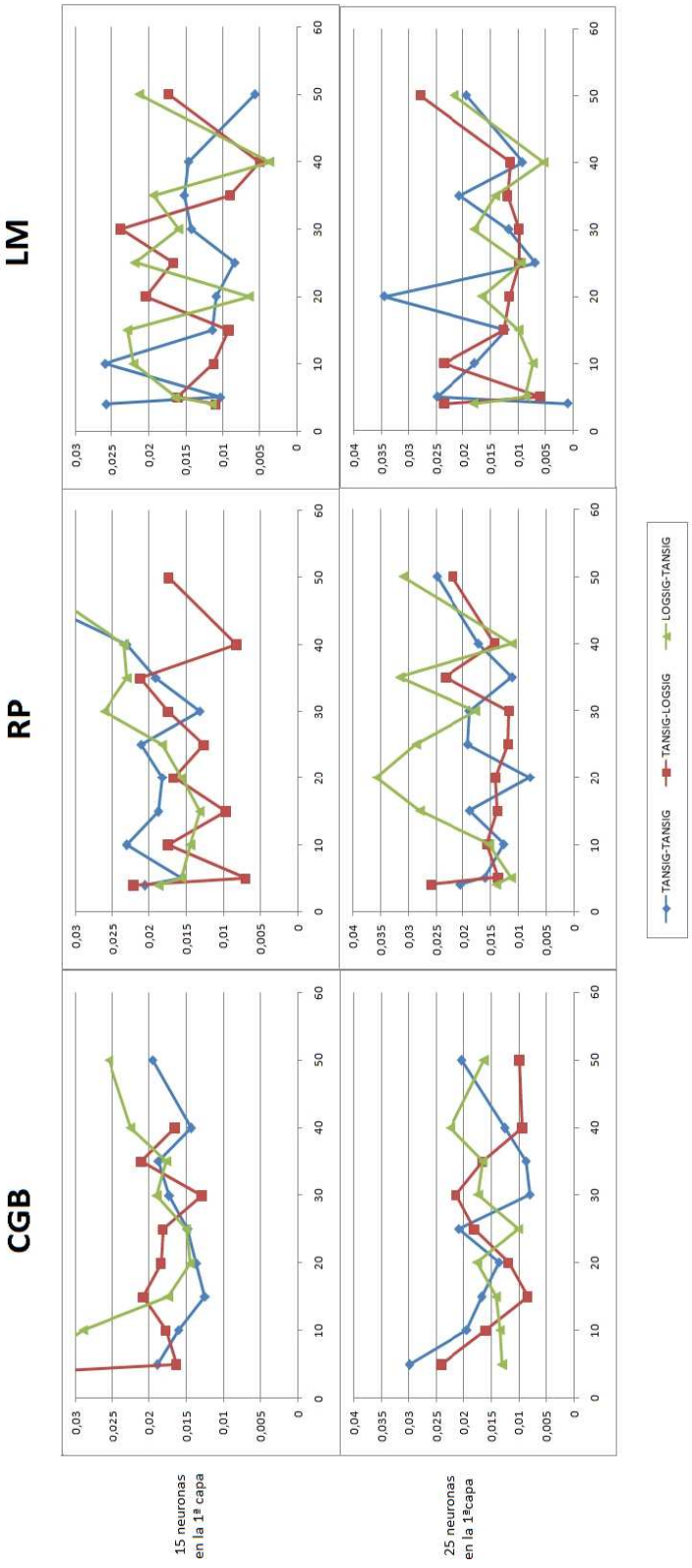

Figura 25: Comparación de los algoritmos con las distintas combinaciones de funciones de activacion, para 15 y 25 neuronas en la segunda capa

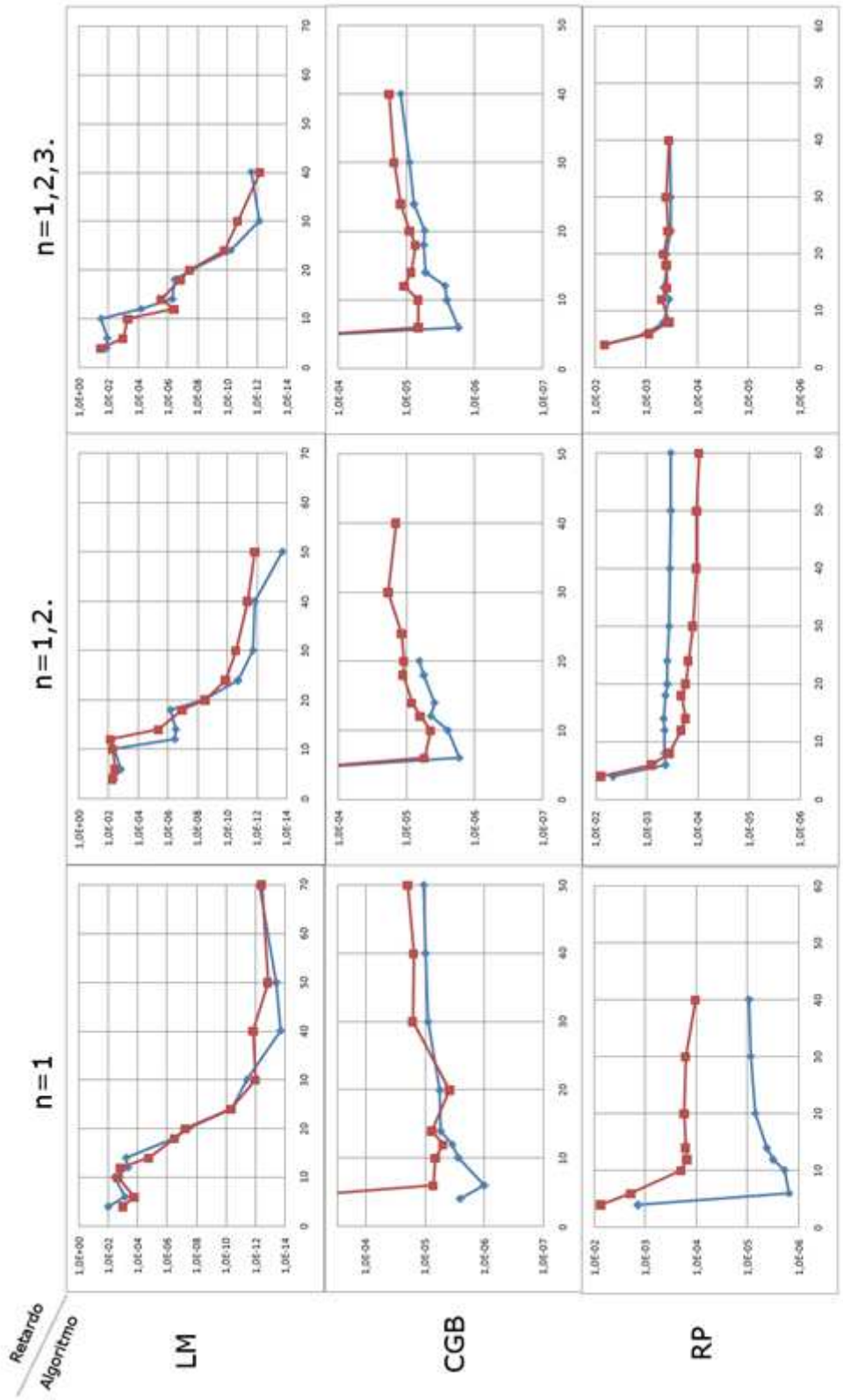

Figura 26 : Comparación de los diferentes algoritmos, retardo y funciones de treansferencia en función del número de neuronas

Algoritmo RP

El aumento del retardo afecta negativamente a este algoritmo. En mayor medida a la función de activación *tansig* que a *logsig*.

Es sin duda al algoritmo que más negativamente afecta la inclusión de memoria.

Red dinámica de dos capas

Se va a observar cómo afecta la topología de dos capas ocultas, fijando el retardo a 1 (ver Figura 27).

A rasgos generales, los resultados son muy parecidos, o incluso peores que la red de una capa y retardo 1 (Figura 26, primera columna). En los tres algoritmos existe la ya clásica zona de inestabilidad debido a la falta de neuronas, y una zona de evolución aproximadamente lineal, ya sea decreciente (LM), creciente (CGB) o uniforme (RP).

Se pasa a evaluar como se ve afectado cada algoritmo.

Algoritmo LM

El aumento de neuronas en la primera capa reduce la zona de inestabilidad, para 16 neuronas son necesarias 20 neuronas en la segunda capa, siendo en cambio necesarias 14 en el caso de tener 30 elementos en la primera capa oculta.

Un mayor número de neuronas en ambas capas favorece a la minimización del error en este algoritmo.

Algoritmo CGB

La inclusión de una segunda capa, mejora los resultados de este algoritmo, el cual alcanza su mínimo después de la zona de inestabilidad. Es curioso que el número de neuronas en la primera capa afecte tan mínimamente a la red, esto permite que no se invierta capacidad computacional en incrementar el número de elementos, facilitando la creación de redes pequeñas y rápidas.

Algoritmo RP

La zona de inestabilidad es la que se ve más afectada por la segunda capa, ya que según se va a amentando el número de elementos, desciende el error. Pero aun con 30 neuronas en la primera capa, no se obtienen mejores resultados que con una única capa de procesado.

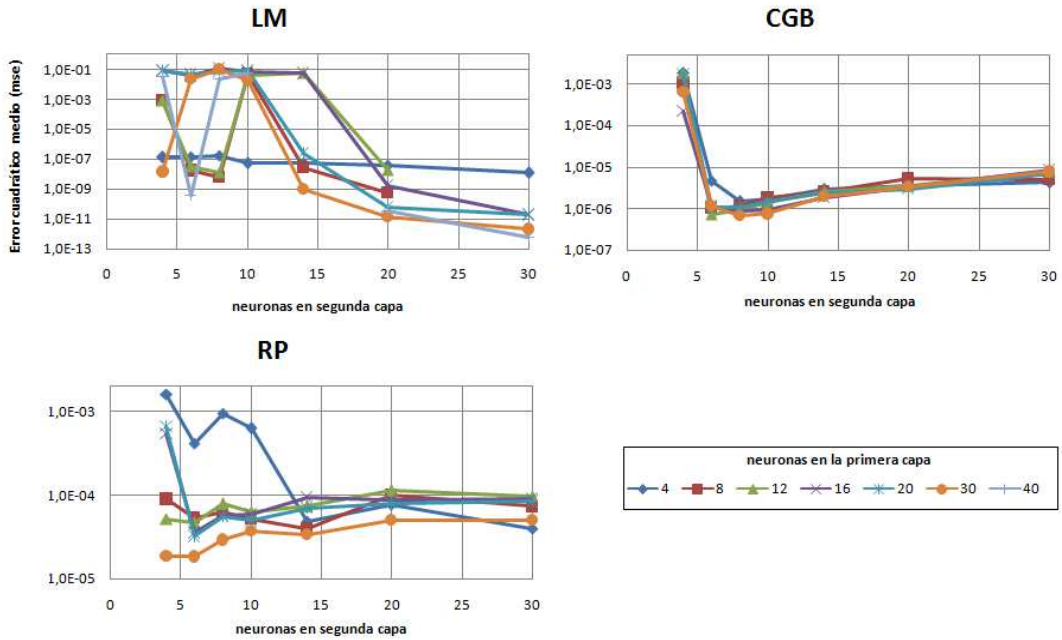

Figura 27: Comparación de combinaciones de parámetros de diseño en redes recurrentes de dos capas.

Metodología de diseño: red dinámica

El objetivo del estudio, era comprender la relación de los distintos parámetros y diseñar un método que minimizara el error. Esto se consigue tomando las decisiones conociendo la influencia que puedan tener entre ellos y sobre la red en general.

Establecer una relación jerárquica es algo difícil ya que en mayor o menor medida todos se afectan entre sí. Pero se puede realizar una aproximación que pueda servir como modelo general, a variar en situaciones particulares.

Esta metodología se ilustra en la Figura 28.

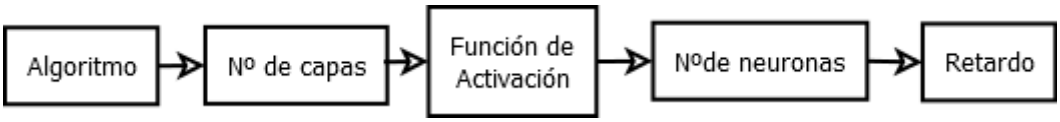

Figura 28: Diagrama de bloques orientativo de la toma de decisiones en el diseño de una red dinámica.

Lo más importante es la elección del algoritmo, ya que de esto dependen las demás decisiones. A continuación se debe elegir el número de capas, lo que abrirá la posibilidad de decidir la función de activación de cada una de las anteriores capas.

Ahora bien, en los algoritmos CGB y RP, lo aconsejable es elegir el número de neuronas previo al retardo, ya que este no afecta al número de neuronas que minimiza el error. Pero para el caso de LM, se debería elegir el retardo previamente al número de neuronas. Como ya se ha comentado, este es un diseño general a particularizar para cada algoritmo.

Estos criterios son los que se van a tener en cuenta a la hora del diseño final de las redes neuronales.

5.4 Diseño de la red estática (*Feed Forward*)

Teniendo ya una idea global del conjunto de parámetros que afectan al diseño de una red neuronal artificial, se pasa al diseño de la red estática.

Este tipo de redes no poseen memoria, por lo que no les es posible distinguir patrones temporales. Esto obliga a extraer características globales de cada golpe, ya sean espectrales o temporales, y que éstas sean suficientemente robustas como para realizar una buena diferenciación de clases.

5.4.1 Características a extraer

Las entradas de la red deben ser características que diferencien un golpe de otro, y éstas entre sí deben estar lo menos relacionadas posible, esto es, que no evalúen una misma propiedad.

Se proponen las siguientes características:

Asimetría (*Skewness*)

Es una medida de la asimetría del espectro con respecto a la media. Si este valor es positivo, implica que la distribución espectral con respecto a la media tiene mayor peso hacia baja frecuencia, y viceversa.

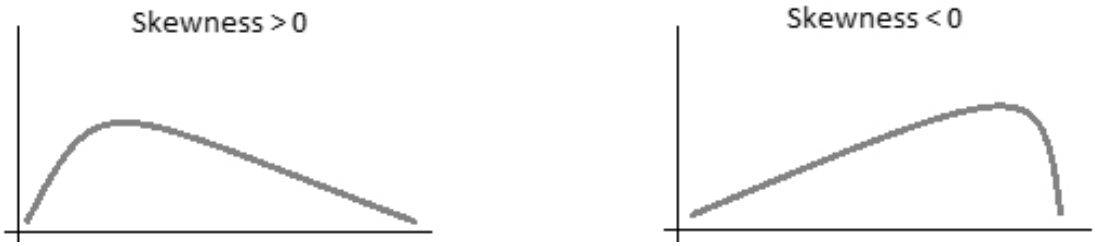

Figura 29: Espectros con variación de asimetría (skewness).

La expresión matemática se define como:

$$skewness = E\left[\left(\frac{X-\mu}{\sigma}\right)^3\right] \quad (16)$$

siendo: E la esperanza, X la variable de entrada, $\boldsymbol{\mu}$ es el momento central y $\boldsymbol{\sigma}$ la desviación estándar.

Por lo tanto esta característica nos informa globalmente de lo grave o agudo que es el sonido de entrada.

Irregularidad del espectro

Esta característica pretende clasificar los espectros según la rapidez de sus variaciones a lo largo del eje frecuencia.

Figura 30: Espectros con variación de la característica irregularidad.

El cálculo se realiza midiendo el número total de desviaciones haciendo uso de las derivadas en cada punto.

Posición del máximo

La localización de la posición del valor máximo del espectro. Esta parece una característica muy útil a la hora de clasificar los patrones sonoros, y es así, pero no en el tipo de señales que nos ocupa, que básicamente se trabaja en régimen transitorio.

Se calcula inicialmente la FFT, y de ésta se extrae la posición del máximo valor del espectro.

Tasa de cruces por cero

Esta característica mide el número de veces que la señal temporal cruza el valor cero. Se define como:

$$zcr = \frac{1}{K-1}\sum_{k=1}^{K-1} \mathrm{C}\{x_k x_{k-1} < 0\} \qquad (17)$$

$$siendo\ \mathrm{C}\{A\} = \begin{cases} 1 & si\ el\ argumento\ es\ verdadero \\ 0 & si\ el\ argumento\ es\ falso \end{cases}$$

k es el instante actual, Kel número total de muestras a considerar y x la señal de la cual se busca el zcr.

Se suele utilizar para distinguir los sonidos sordos de los sonoros, pero dado que las señales que nos ocupan son todas sordas, lo que realmente nos dará es una idea de la orientación frecuencia del sonido en la más original de sus características: si una señal cruza muchas veces por cero, será de más alta frecuencia que una que lo hace poco.

Centroide

Representando el espectro como una figura bidimensional, su centroide se corresponderá a la intersección de todos los hiperplanos de ésta. Informalmente, corresponde a la media aritmética de todos los puntos de espectro.

Siendo el espectro $X(k)$, K_+ el conjunto de valores positivos y k el índice de frecuencia. El centroide espectral vendrá dado por la expresión:

$$C_f = \sum_{k \in K_+} kX(k) \tag{18}$$

Como característica añadida, también se propone el centroide temporal, calculado análogamente al espectral, pero en base a la envolvente.

Curtosis

Esta medida evalúa la concentración de valores que se encuentran cerca de la media (en este caso el centroide). Una mayor curtosis implica un mayor número de valores espectrales cerca de C_f.

Siendo E la esperanza, $X(k)$ un punto dado del espectro, C_f el centroide y σ la desviación estándar. La curtosis viene dada por la expresión:

$$Curtosis = \frac{E\left[X(k) - C_f\right]^4}{\{E\left[X(k) - C_f\right]^2\}^2} \tag{19}$$

Coeficientes espectrales de la escala de Mel (MFCCs)

Este método basado en el cepstrum (ver explicación teórica pagina), pero con un eje de frecuencias no lineal, siendo este eje elegido teniendo en cuenta los criterios psicoacústicos del oído humano.

El objetivo es encontrar descriptores del espectro que tengan que ver con la forma en la que oímos.

Para obtener los MFCCs es necesario crear un banco de filtros de la escala de Mel. se construye equiparando un tono de 1000Hz y a 40dBs por encima del umbral de audición del oyente, con un tono de 1000mels. Este banco de filtros tiene un diseño triangular solapado al 50% y con un ancho de banda y separación determinada por un intervalo Mel constante (uniformemente equiespaciados).

La relación entre frecuencias Mel y frecuencia en Hz se calcula mediante la relación de la ecuación (20), donde f denota la frecuencia real y $mel(f)$ la frecuencia percibida.

$$mel(f) = 2595\, x \log_{10}\left(1 + \frac{f}{700}\right), \quad \text{ó} \quad mel(f) = 1125\, x\, ln\left(1 + \frac{f}{700}\right) \tag{20}$$

Los MFCCs se calculan de la siguiente forma:

1. Calculo de la FFT: Se calcula trasformada rápida de Fourier al tramo bajo estudio.

2. Transformación a escala Mel: se filtra el espectro con el banco de filtros descrito con anterioridad.

3. Calculo de la DCT: se transforma al dominio del tiempo mediante la transformada discreta del coseno, la cual representa la parte real de la transformada inversa de Fourier

De esta forma ya se tienen los MFCCs.

En la aplicación que nos ocupa, se han tomado 20 coeficientes cepstrales de la escala de Mel.

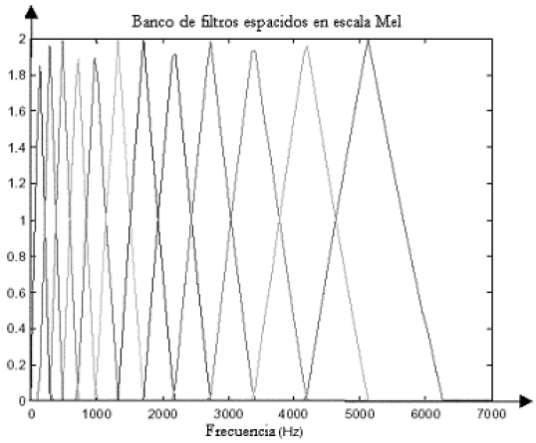

Figura 31: banco de filtros triangulares de la escala de Mel.

5.4.2 Selección automática de características

Dadas la infinitas posibilidades de superficies de las que el usuario puede hacer uso, cabe la situación de que una característica diferencie perfectamente los golpes en una situación, y en cambio tenga el mismo valor para otra. En este último caso, si la características es utilizada como entrada para el entrenamiento de la red neuronal, ésta mal-aprenderá que dos golpes tienen los mismos valores. (ver explicación teórica página 26). Además de que el cálculo del conjunto de características que se han propuesto suman un total de 26, lo cual supone un alto coste de procesado.

Algoritmo inter clase

Se ha implementado un sencillo algoritmo de comparación, basado en las diferencias entre la media de valores de una característica para dos clases diferentes.

Se toma una característica, y se calcula la media de los valores de todos los golpes grabados para cada clase. Las clases son comparadas dos a dos. Se pasa a explicar un ejemplo:

Teniendo una característica a evaluar entre dos clases a clase A y clase B, siendo la media de B menor que A. Se establece un umbral para contabilizar el número de golpes de B que lo supera tal que

$$umbral = minimo_A + (media_A - minimo_A) \cdot \varphi \qquad (21)$$

siendo $minimo_A$ el menor valor de la clase A, $media_A$ el valor medio de todos los golpes de A y φ una constante que variará el umbral, siendo éste igual al $minimo_A$ (para $\varphi = 0$) y tomando el valor $media_A$ para $\varphi = 1$.

Como se puede observar gráficamente en la Figura 32, un valor (golpe 17) de los 40 supera el umbral, por lo que un 97.5% de los valores de la clase B se encuentran por debajo del umbral.
Esto no quiere decir que exista ese porcentaje diferencia entre ambas clases, ya que si se observa el golpe 6 de la clase A, se ve que está incluso por debajo del mínimo valor de B. Es simplemente un indicador de lo diferentes que son los valores de esta característica, para su posterior evaluación.

Las clases son evaluadas dos a dos para cada una de las características, obteniendo una tabla de comparación para cada una de ellas.

Finalmente, la suma de los índices de comparación de cada característica son evaluados, tomándose el número de características que mayor índice de diferenciación posean (5 por ejemplo).

El problema de este algoritmo es que no evalúa con precisión la variabilidad de una misma clase.

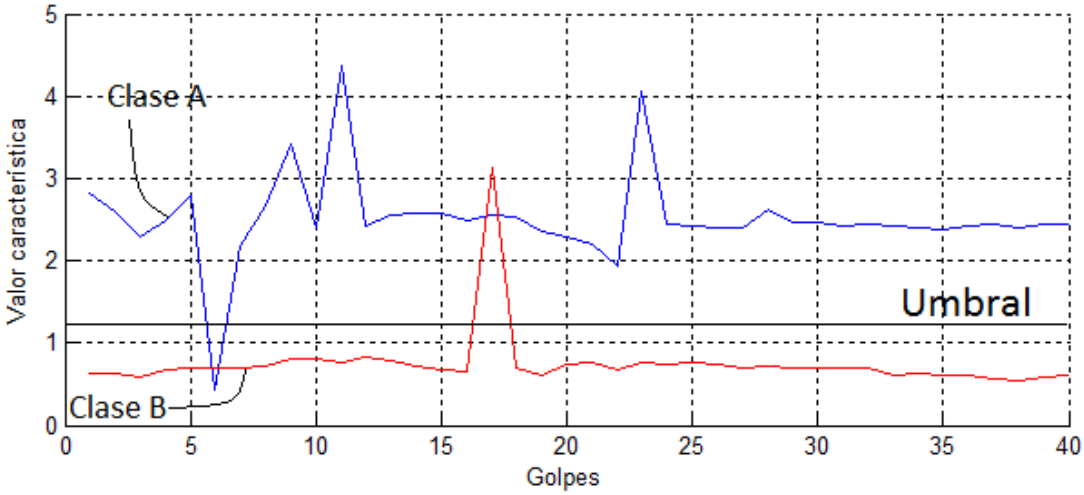

Figura 32: Ejemplo de evaluación de una característica para dos clases.

Algoritmo intra-inter clase (RELIEF)

Dentro de los algoritmos de selección de características, RELIEF [42] es considerado uno de los más exitosos dada su simplicidad y eficiencia [43].

Se pasa a dar una explicación matemática. Dado un conjunto D tal que

$$D = \{(x_n, y_n)\} \tag{22}$$

donde x_n es la muestra n-ésima y y_n la clase a la que corresponde, la idea de RELIEF es estimar de manera iterativa el peso de la característica de acuerdo con su habilidad de discriminar entre patrones vecinos. En cada iteración, un patrón x es seleccionado aleatoriamente así como sus dos patrones vecinos, uno de su misma clase (NH), y otro de una de las clases opuestas (NM).

El peso de la i-esima característica es entonces actualizado según la expresión

$$w_i = w_i + \left|x^{(i)} - NM^{(i)}(x)\right| - \left|x^{(i)} - NH^{(i)}(x)\right| \tag{23}$$

de esta forma se obtiene un peso o ponderación para cada una de las características bajo estudio.

Esto permite establecer un umbral de importancia en la clasificación, y elegir todas aquellas características que lo superen, asegurando una buena discriminación inter e intra clase.

Los resultados de la comparación entre los algoritmos anteriormente explicados hacen elegir RELIEF como método de clasificación, ya que además de realizar un análisis más profundo, permite la ponderación de las características pudiendo ser éstas elegidas por su calidad de clasificación y no escoger un número fijo independientemente de su validez como clasificadores.

5.4.3 Diseño específico

Siguiendo la metodología de diseño estudiada previamente se han elegido redes de dos capas ocultas por ser más eficientes que las de una capa.

Al testar una misma red con varios sets de inputs, se ha observado que el número de neuronas que minimiza el error varía en cada caso.

A efectos prácticos, esto se traduce en que para cada combinación de superficies que se elijan (cada usuario elegirá un set especifico) habrá una red que realice mejor la tarea.

En la Tabla 2, se ha fijado el número de neuronas en la primera capa, y buscado los elementos que menor error ofrezca a la salida.

Se entiende entonces que diseñar una red única que satisfaga todas las situaciones es una tarea imposible.

La solución que se propone parte del mismo principio que en la elección dinámica de las inputs. Pero en primer lugar se definirá una red de base, a la que se llamará red por defecto.

Elección de la red por defecto

El algoritmo de entrenamiento que mejores resultados tiene es LM (ver Tabla 2), con 14 neuronas en la primera capa. Pero antes de elegir esta red como definitiva hemos de evaluar los tiempos de entrenamiento y procesado. En la Tabla 3 se observa el tiempo medio que se tarda en entrenar una red fijado el número de épocas (entrenamiento), y el tiempo medio que ésta tarda en procesar los datos hasta obtener la salida (procesado). Este último se obtendrá

midiendo el tiempo que cada red invierte en tomar la decisión promediando para un total de 50 sets de datos de entrada.

Tabla 2: Comparación del error medio para las diferentes topologías (fijándose el número de neuronas en la primera capa) de cada algoritmo para 4 sets de datos diferentes.

	Algoritmo	RP	CGB	LM
SET 1	error	0,01570	0,01691	0,00862
	neuronas por capa	20/7	22/10	14/12
SET 2	error	20/15	0,00743	0,00633
	neuronas por capa	20/15	22/5	14/7
SET 3	error	0,00962	0,00939	0,00754
	neuronas por capa	20/26	22/6	14/7
SET 4	error	0,00834	0,01861	0,00629
	neuronas por capa	20/26	22/9	14/22
Error medio		**0,00993**	**0,01309**	**0,00720**

Tabla 3: Comparación de tiempos de procesado y entrenamiento para los tres algoritmos considerados.

Algoritmo	Entrenamiento (s)	Procesado (ms)
RP	0,513	6,67
CGB	0,963	6,71
LM	0,929	6,70

El algoritmo que requiere menor tiempo de entrenamiento es RP, siendo casi el doble de rápido que los demás. Esta diferencia afecta al tiempo que el usuario deberá esperar mientras se realiza la selección dinámica. Poniendo un ejemplo, si se desean probar 15 redes, haciendo 10 pruebas con cada una para obtener la media del error cuadrático medio, el tiempo de espera para cada red será:

$$T_{RP} = 15x10x0.513 = 76\ s = 1'16''$$
$$T_{CGB} = 15x10x0.963 = 144.45\ s = 2'24''$$
$$T_{LM} = 15x10x0.929 = 2'19''$$

El tiempo de procesado que es requerido por los distintos algoritmos no alberga diferencias significativas, estando cerca de los 7ms. Por lo que se vio en la revisión de conocimientos de cualquier algoritmo en tiempo real, la ventana de integración del oído esta en torno a los 40ms, por lo que 7ms consume aproximadamente la sexta parte del tiempo del que se dispone.

Por los resultados en cuanto al error se va a elegir la red de 14 neuronas en la primera capa y 7 en la segunda, entrenada con LM, siendo esta topología la que globalmente tiene menor error. Las funciones de transferencia elegidas son: Logsig en la primera capa, tansig en la segunda y la función lineal en la capa de salida.

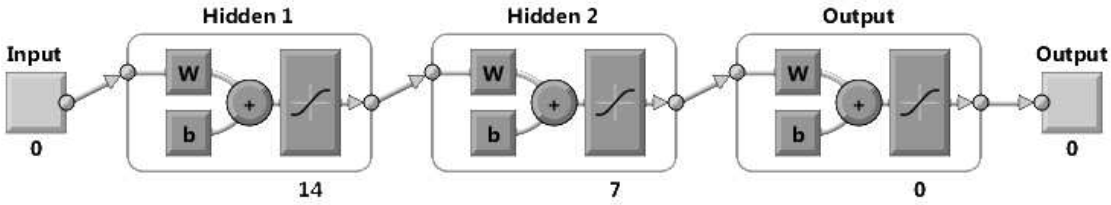

Figura 33: Red neuronal estática, que el sistema entrena por defecto si no se modifica la calidad de ésta.

Selección dinámica del número de neuronas

Se trata de entrenar varias redes y elegir la que menor error obtenga. Es evidente que esto supondrá un aumento del tiempo de espera en el aprendizaje de la red, por lo que se permite al usuario, por medio de la interfaz grafica, la posibilidad de habilitar esta opción o dejar la red creada por defecto.

Idealmente, se deberían comprobar todas las combinaciones de neuronas en ambas capas, pero esto aumenta excesivamente el tiempo que se dedica al entrenamiento. Fijar el número de neuronas en la primera capa apenas empobrece la interpretación de la red y disminuye notablemente el número de redes a evaluar. Esto se puede observar en la Tabla 2, donde se ha fijado el número de elementos en la primera capa.

Si el usuario habilita esta función, se entrenarán replicas de la red por defecto variando el número de neuronas en la segunda capa. La que obtenga menor error, será la red a entrenar finalmente.

Tabla 4: Grados de calidad en la optimización de la red estática.

	Mayor calidad				Menor calidad			
Distancia entre neuronas	1	1	2	2	3	3	4	4
Número de repeticiones	14	12	10	8	8	7	7	6
Tiempo de optimización(s)	325,1	278,7	116,1	92,9	61,9	54,2	40,6	27,9

El problema observado del excesivo tiempo de aprendizaje se soluciona permitiendo al usuario decidir la "calidad" de la optimización de la red. Se proponen 8 grados de calidad de optimización de la red, teniendo en cuenta dos criterios: número de veces que se va a entrenar una misma red para el cálculo medio del mse, y cada cuantas neuronas se entrena.

Se van a considerar de 5 a 30 neuronas, tomadas de 1,2,3 y 4. Pudiendo variarse entre 6 y 14 repeticiones de entrenamiento por red.

5.5 Diseño de la red recurrente

Teniendo ya una visión general de la metodología de diseño a seguir. Lejos de tomar las decisiones en una única dirección, es decir, elegir un algoritmo luego el número de capas etc., se van a considerar varias posibilidades en cada decisión, y las mejores combinaciones se evaluarán finalmente.

5.5.1 Elección del tramo de entrada

Como entrada, esta red va a tener los golpes interpretados por el usuario. Estas señales tendrán una duración que variará dependiendo de la superficie golpeada, características físicas como son el volumen, la resonancia o la masa son determinantes. Pero no es la totalidad de la señal percusiva lo que importa, será la variación temporal lo que se tendrá en cuenta en este caso.

Ya que el objetivo es encontrar la mayor diferencia entre los golpes, existirán ciertas partes de la señal temporal que serán de mayor interés que otras.

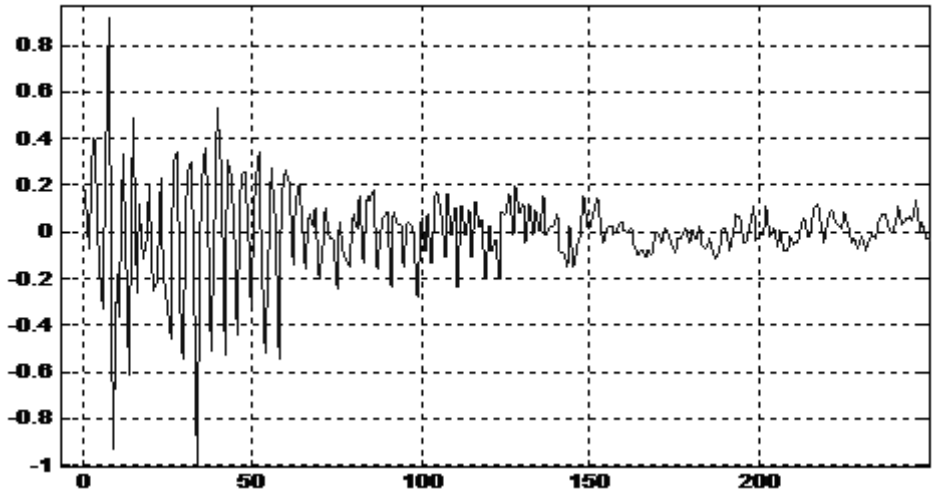

Figura 34: Señal captada del sonido producido por impacto de un lápiz contra una mesa.

En la Figura 34 se observa la señal producida por un lápiz al golpearse su extremo contra una mesa. Las primeras 50 muestras aproximadamente (ataque) poseen información del elemento de excitación y del elemento excitado, en cambio a partir de ahí, hasta 150 es el sistema excitado el único que se encuentra radiando y de 150 en adelante, la mesa se encuentra en régimen libre.

La zona que mayor interés tiene en cuanto a la distinción es el ataque, ¿pero cuantas muestras son las optimas?. Como es habitual en el desarrollo de de este proyecto, no existe la posibilidad de determinar un valor único dadas las infinitas posibilidades de superficies y elementos de excitación de los que el usuario puede servirse. Pero asumiendo esto, se puede llegar a una solución general que satisfaga a grandes rasgos todas las situaciones.

En la Figura 35 se muestra la evolución del error medio frente al número de neuronas en una red de Elman de una capa, para 3 sets de datos diferentes.
Se puede observar que para cada set, y para cada topología existe un número de muestras que optimiza la red.

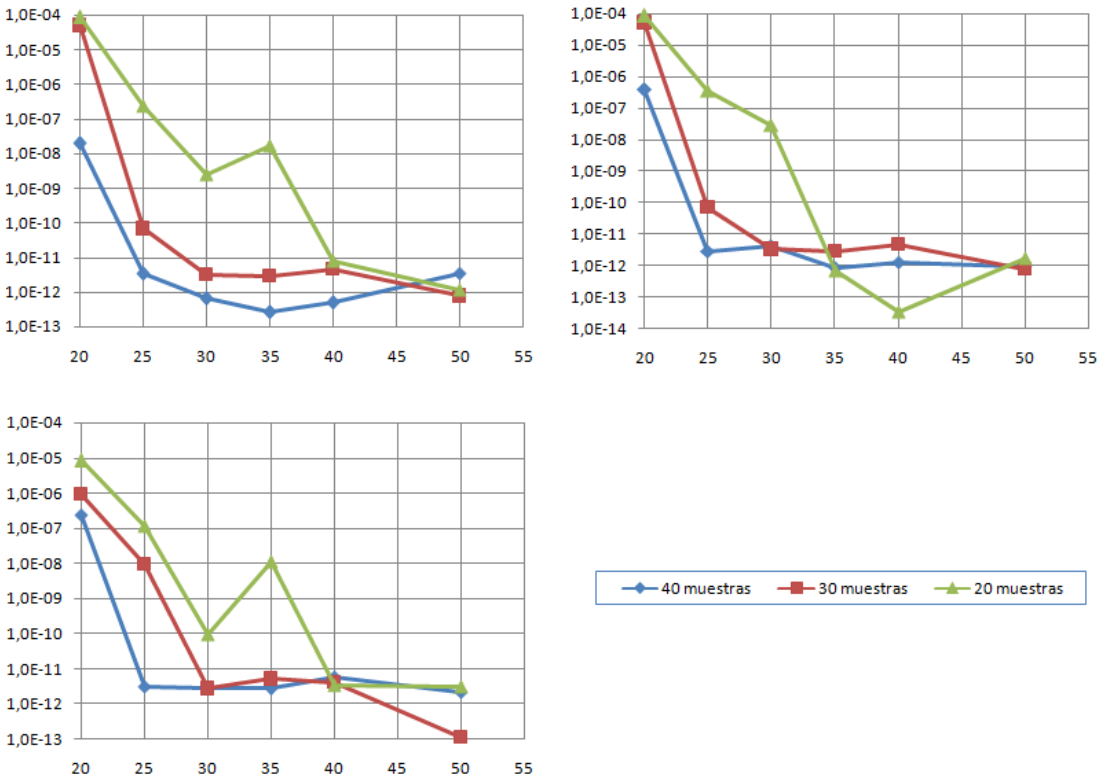

Figura 35: Error medio frente al número de neuronas, para 3 sets de datos diferentes (uno en cada gráfica) y 3 posibilidades de muestras a la entrada (ver leyenda).

Esto obliga a tener en cuenta el número de muestras en la entrada como otro parámetro más de diseño de la red neuronal. De la Figura 35 se extrae que 40 muestras es la situación más estable para las tres situaciones, manteniendo un valor bajo de error asegurando una zona plana para el número de neuronas de interés, por lo que se va a comenzar el diseño de la red con 40 muestras.

5.5.2 Posibles redes recurrentes

Siguiendo la metodología de diseño estudiada anteriormente. La elección del algoritmo de entrenamiento condiciona la mayor parte de los parámetros de diseño. A la hora de tomar esta decisión hay que observar dos cuestiones, la minimización del error cuadrático medio, y el tiempo de entrenamiento.

De forma colateral, el algoritmo implica la búsqueda del número de neuronas y capas que minimizan el error. Luego a la hora de hacer uso de la red neuronal en tiempo real, existirán topologías que sean más rápidas que otras.

Teniendo en cuenta lo expuesto por Hagan [33], y lo evaluado en la metodología de diseño:

- LM: posee bajo error, pero alto coste computacional y de memoria. Pudiendo llegar a superar los límites permitidos por MATLAB.
- RP: entrena la red rápidamente, dando peores resultados que LM, pero posibilita la creación de redes neuronales más rápidas.
- CGB: un método basado en el gradiente conjugado, que representa un estadio intermedio entre la velocidad de RP y el bajo error de LM.

Se va a diseñar una red para cada algoritmo y, comprobándose los resultados en cuanto a calidad de detección y tiempo de entrenamiento, se elegirá la red.

Se han diseñado tres redes

- LM: Red de una capa realimentada. 30 neuronas en la capa oculta.

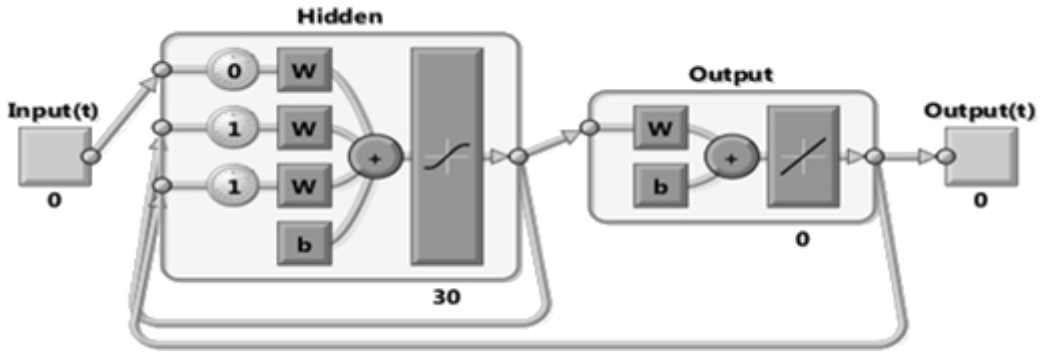

Figura 36: RNA recurrente propuesta con entrenamiento LM.

- RP: red de dos capas, 30 neuronas en la primera y 12 en la segunda. Con lazos propios en cada capa e interconectadas entre sí.

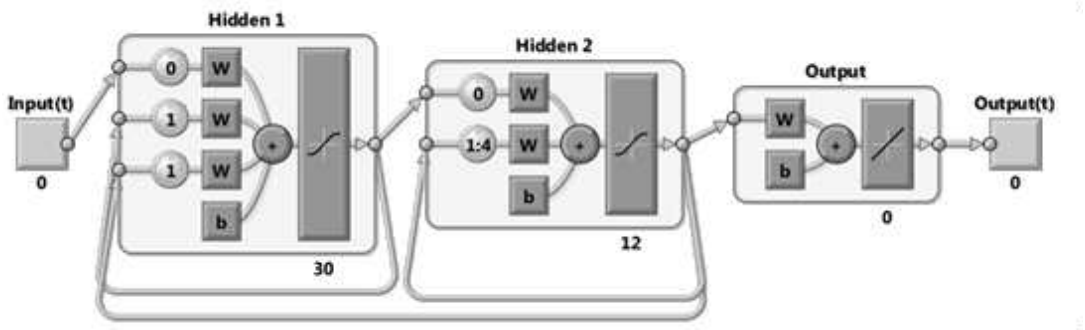

Figura 37: RNA recurrente propuesta con entrenamiento RP.

- CGB: Red de dos capas ocultas. Podría considerarse la unión de dos redes en cascada. Una red recurrente de Elman constituiría la primera capa de 8 neuronas, y una red de propagación hacia adelante (Feed-Forward) en la segunda capa con 18 neuronas.

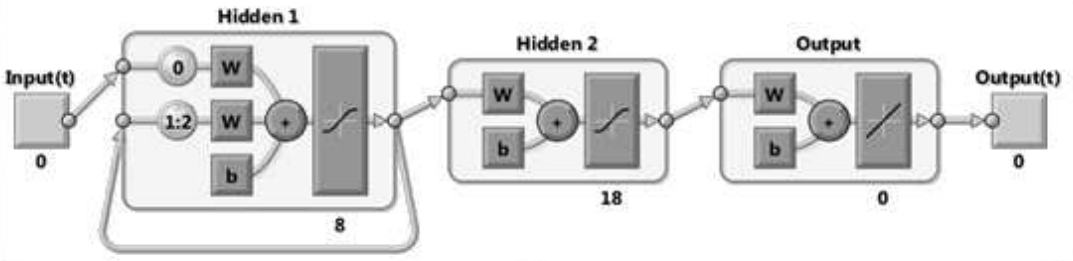

Figura 38: RNA recurrente propuesta con entrenamiento CGB.

Condiciones de fin de entrenamiento

Como se comentó en los fundamentos de redes neuronales, existen varias maneras de detener el entrenamiento. Estos parámetros de detección se han configurado para finalizar cuando:

- el valor absoluto del gradiente sea menor a 10^{-8}
- se sobrepasen 10 *checks* de validación
- se supere 1min 20 segundos de tiempo de entrenamiento
- se superen 500 iteraciones

En la fase de entrenamiento, el conjunto de datos es dividido para las distintas fases (ver fundamentos pagina 41). Lo más común es dividir estos datos aleatoriamente, pero para

comparar los resultados de las redes, esto obligaría a entrenar las redes un gran número de veces para eliminar el factor azar. Por ello se va a realizar la misma división en todos los sets de datos.
Se partirá de 40 golpes por cada clase. 10 golpes de cada (40 en total) constituirán la fase de test. Esto implica que cada error tendrá un peso del 2.5%.

Evaluación de las redes neuronales

Con las condiciones de entrenamiento mencionadas con anterioridad, se pasa a evaluar el funcionamiento de las tres redes propuestas. Se hace uso de una base de datos de diferentes combinaciones de golpes en estructuras como: mesas de diferentes tipos, cajas, vasos, hasta una lámpara; siendo golpeados con lapiceros, cucharas, baquetas y las manos entre otros elementos.

Los resultados de error cuadrático medio (*mse*) tiempo de entrenamiento, y el porcentaje de acierto obtenido en la fase de test se reflejan en la Tabla 5.

LM es sin duda la red que menor error tiene. Al disminuir su error rápidamente, logra superar la condición de gradiente mínimo, finalizando su entrenamiento en 22 segundos como máximo.
No es así para RP, ya que en 4 ocasiones, su entrenamiento finalizo por exceder el tiempo máximo al no descender el gradiente menos de los establecido, ni superar los 10 intentos de minimizar el error.

Se elije por tanto la red LM como red recurrente. Más de un 95% es un valor aceptable. Hay que tener en cuenta que la señal de la que se parte es de origen temporal, y por ende muy susceptible a pequeñas variaciones, por lo que este valor puede descender dependiendo de la habilidad del usuario de producir la misma señal. Siendo estas variaciones producidas por las distintas formas de golpear la superficie algo inevitable, se considera un 95.3% de acierto es un buen resultado.

5.6 Conclusiones

En este capítulo se ha abordado el problema de la transcripción de la señal de audio correspondiente a los impactos que el usuario interprete sobre objetos de su entorno.

Tabla 5: Comparación de las 3 diferentes redes neuronales propuestas.

	MSE			Tiempo de entrenamiento (s)			Aciertos en fase de test (%)		
	LM	CGB	RP	LM	CGB	RP	LM	CGB	RP
SET 1	3,80E-26	0,02560	0,00930	21	17	37	90	87,5	87,5
SET 2	8,52E-30	0,01170	0,00137	22	23	80	100	100	100
SET 3	4,43E-20	0,00302	0,00878	22	22	25	97,5	90	85
SET 4	5,25E-20	0,00013	0,00236	17	80	39	87,5	87,5	90
SET 5	1,90E-26	0,00214	0,00102	16	43	46	100	87,5	92,5
SET 6	8,42E-23	0,00211	0,00037	16	26	80	97,5	92,5	97,5
SET 7	1,83E-19	0,00525	0,00094	17	21	80	95	97,5	95
SET 8	2,03E-23	0,00616	0,00239	22	37	64	92,5	95	92,5
SET 9	7,92E-21	0,00537	0,00833	17	20	20	95	97,5	97,5
SET 10	4,61E-31	0,00895	0,00113	20	28	80	97,5	87,5	85
MEDIA	**2,88E-20**	**7,04E-03**	**3,60E-03**	**19**	**31,7**	**55,1**	**95,3**	**92,3**	**92,3**

Partiendo de una visión global de la disposición física de los elementos se ha procedido a describir el proceso de configuración, que consistirá en la grabación de las distintas clases de impactos. De estos datos grabados se pasará a la detección de eventos mediante una maquina de estados y la posterior extracción de características.

Se han detallado las propiedades de estas diferentes características tanto espectrales como temporales y la enorme importancia de un algoritmo de selección de características inter e intra clase como es RELIEF.

Antes de proceder al diseño especifico de las redes neuronales, se ha observado cómo afectan los diferentes parámetros a la red, y la forma en la que éstos se interrelacionan. Los resultados de la experimentación con redes tanto dinámicas como estáticas de hasta dos capas ocultas, concluye en una metodología orientativa de diseño, estableciendo un orden jerárquico de importancia en la toma de decisiones. Se destaca la importancia de la elección del algoritmo de entrenamiento para la optimización de los resultados de la red.

También es importante dejar claro que el aumento de neuronas o capas de forma excesiva, incrementa el error. Por lo general, la función de error medio con respecto al número de neuronas es alto para un deficiente número de elementos (además de inestable) y conforme se va aumentando, el error va disminuyendo hasta llegar al mínimo absoluto, a partir del cual, el error comienza a aumentar.

Por lo que existe un numero de neuronas y capas que maximiza la eficiencia del sistema. Pero los resultados obtenidos también muestran que estos parámetros varían según los objetos elegidos. Es decir, una red será muy eficiente si golpeamos ciertos elementos, y tendrá un alto error para otros.

El diseño de la red estática concluye en la topología de dos capas ocultas con 14 y 7 neuronas respectivamente y la combinación de funciones de activación tansigmoidal-logsigmoidal.
Para paliar la variabilidad de eficiencia de la red al cambiar los objetos a golpear, descrita con anterioridad, se propone una solución basada en la selección automática de neuronas. El sistema entrena varias redes neuronales variando el número de elementos en la segunda capa. Eligiéndose la red con menor error cuadrático medio se asegura una alta eficiencia para cada combinación de objetos que pueda elegir el usuario. El numero de diferentes redes a evaluar depende del grado de optimización que se elija por medio de la interfaz gráfica.

La red dinámica por otra parte, tendrá como entrada un tramo del ataque la señal temporal (40 muestras). El diseño final cuenta con una única capa oculta de 30 neuronas. El alto tiempo de entrenamiento de esta red no permite el proceso de optimización descrito.

Los resultados de esta red son más pobres que en el caso de la topología estática, esto es debido a las pequeñas variaciones temporales que puedan sucederse en la interpretación de los impactos.

Ambas redes han sido entrenadas con el algoritmo de Levenberg-Marquardt (LM), demostrándose su eficiencia en la minimización de la función de error.

Capítulo 6

Entorno gráfico

El entorno grafico representa la comunicación entre los sistemas de transcripción y el usuario.

El proyecto que se presenta consta de dos partes, la transcripción melódica y la rítmica. Ambas poseen su propia interfaz conectada por el llamado "Menú Principal". La jerarquía de las interfaces graficas se muestra en la Figura 39.

Se pasa a detallar la las características de cada una de ellas.

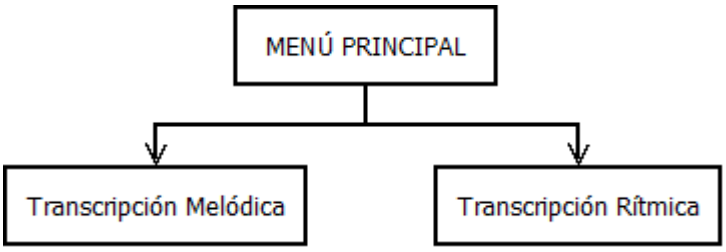

Figura 39: diagrama jerárquico de las diferentes GUIs.

6.1 GUI: Menú Principal

Su cometido es unir las dos partes del proyecto. También se informa brevemente del cometido del sistema, y de la autoría del programa (ver Figura 40).

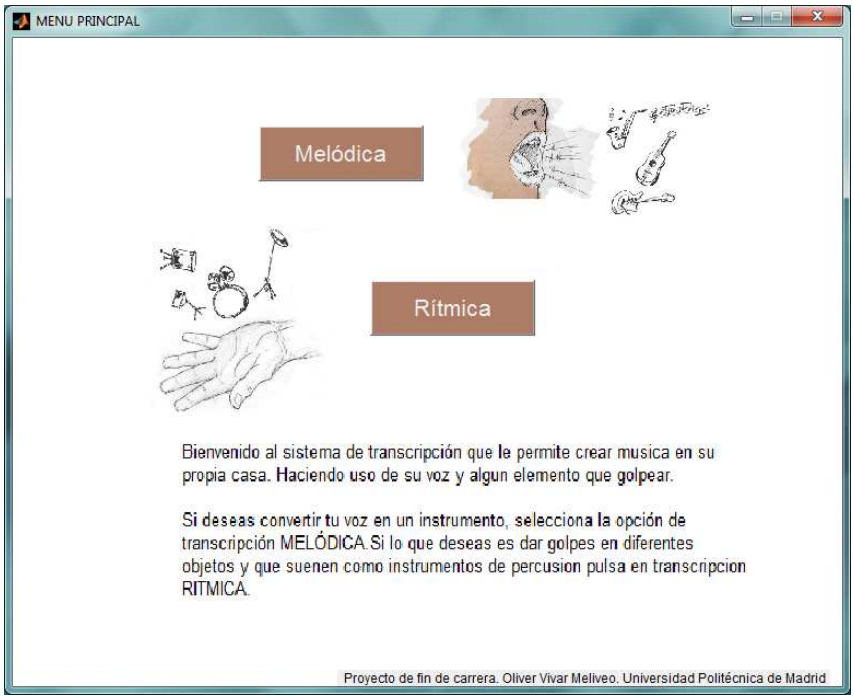

Figura 40: GUI del menú principal.

6.2 GUI: Transcripción melódica

Permite al usuario la configuración de la transcripción melódica, así como variar los parámetros de interpretación y detección.

La GUI (interfaz grafica de usuario), se inicia con algunos botones inactivos (ver Figura 41). El botón de "INICIAR" por ejemplo, se activará cuando se seleccione la fuente de datos, ya sean provenientes del micrófono, o de una señal ya grabada.

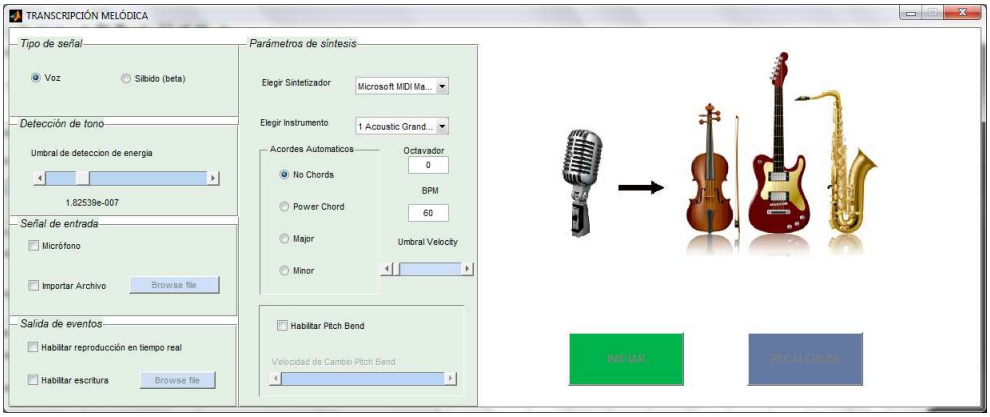

Figura 41: Interfaz de transcripción melódica al iniciarse.

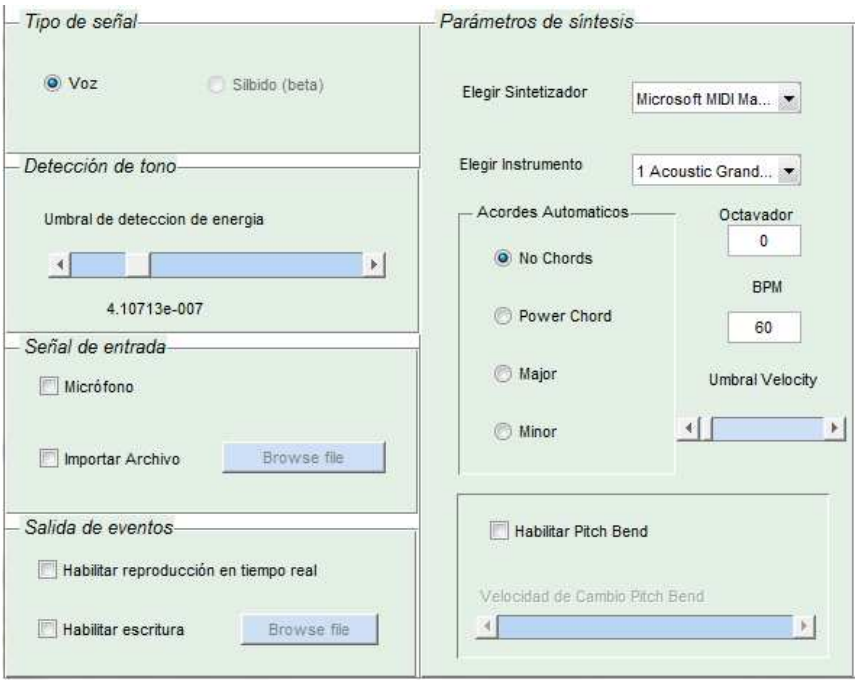

Figura 42: Sección de controles de la GUI melódica.

Descripción de los elementos

Se pasa a describir cada uno de los diferentes elementos con los que el usuario puede interactuar, de arriba abajo y de izquierda a derecha de la Figura 42.

Tipo de señal

Esta parte está dedicada a la elección de la fuente de audio a procesar. La voz (seleccionada en la Figura 42) y la opción de utilizar silbidos, la cual es parte de una próxima etapa de ampliación que permita la selección de una mayor numero de fuentes de audio que se encuentra en fase de desarrollo.

Detección de tono

Al iniciarse la aplicación melódica, se realiza una grabación de ruido ambiente. Este ruido ambiente es almacenado como umbral de detección. Ahora bien, el ruido ambiente puede variar, o al importarse un archivo no puede conocerse el ruido que este contiene. Esto entorpece considerablemente la eficiencia del sistema. Por este motivo se incluye una barra de desplazamiento mediante la cual el usuario puede variar dicho umbral.

Esta posibilidad es muy útil cuando se está usando el sistema en modo *offline*, esto es, cantar al micrófono procesar la señal, y guardarla en un fichero MIDI (.mid). En este modo, la imagen mostrada en la Figura 41 es sustituida por una grafica de la transcripción (Figura 43), el usuario puede aumentar el umbral de detección, y eliminar esos errores (Figura 44).

Señal de entrada

En esta sección se da la posibilidad de procesar la voz proveniente de un micrófono contado a la tarjeta de sonido del ordenador, o de analizar un archivo ya grabado.

Si se activa "Importar Archivo", se habilita la opción ("Browse file") de navegar por el sistema para seleccionarlo. Los archivos deben ser .wav y con frecuencia de muestreo de 8000 Hz.

Salida de eventos

Es donde se elige si deseamos que los mensajes MIDI se manden directamente al sintetizador, o si por el contrario se quiere crear un archivo. En el último caso, se permite la navegación por el sistema mediante el botón "Browse File".

Dependiendo de la señal de entrada elegida, se habilita un tipo de salida de datos.

- Micrófono: Permite tanto el almacenamiento en fichero, como la reproducción en tiempo real.
- Archivo: No permite la opción de reproducción en tiempo real.

Parámetros de síntesis

Aquí se controlan las variables de la interpretación. Desde qué sintetizador utilizar, hasta los BPM en los que se va a grabar el archivo. Se pasa a describir los diferentes campos:

- Elegir sintetizador: Menú tipo *pop-up* que muestra los sintetizadores de los que se puede hacer uso. El sistema no está testado en todos los sintetizadores, por lo que se sugiere usar el sintetizador de Microsoft incluido en todos los PC, "Microsoft MIDI Mapper".

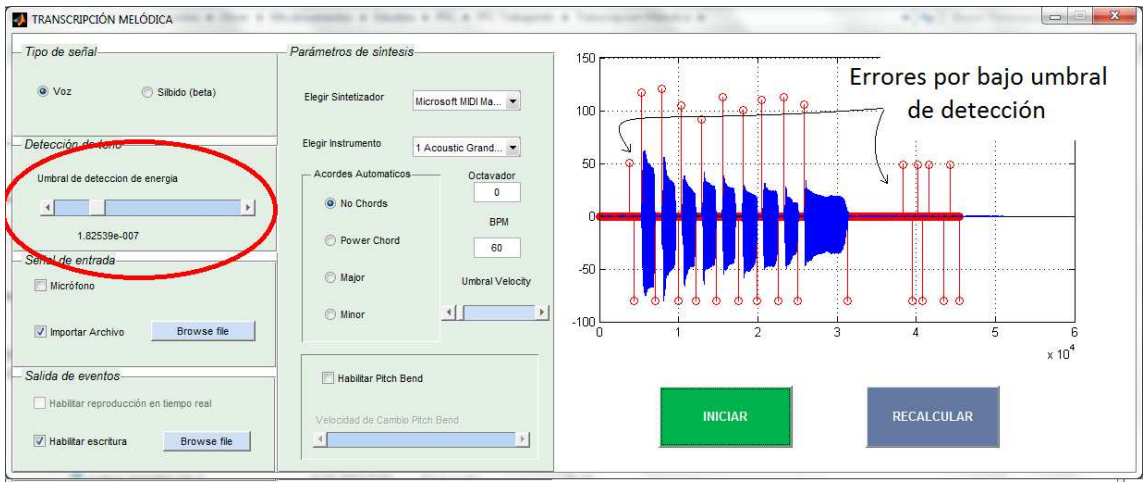

Figura 43: Interfaz de transcripción melódica en modo *offline*. Se detallan los errores de detección y el umbral de detección de energía.

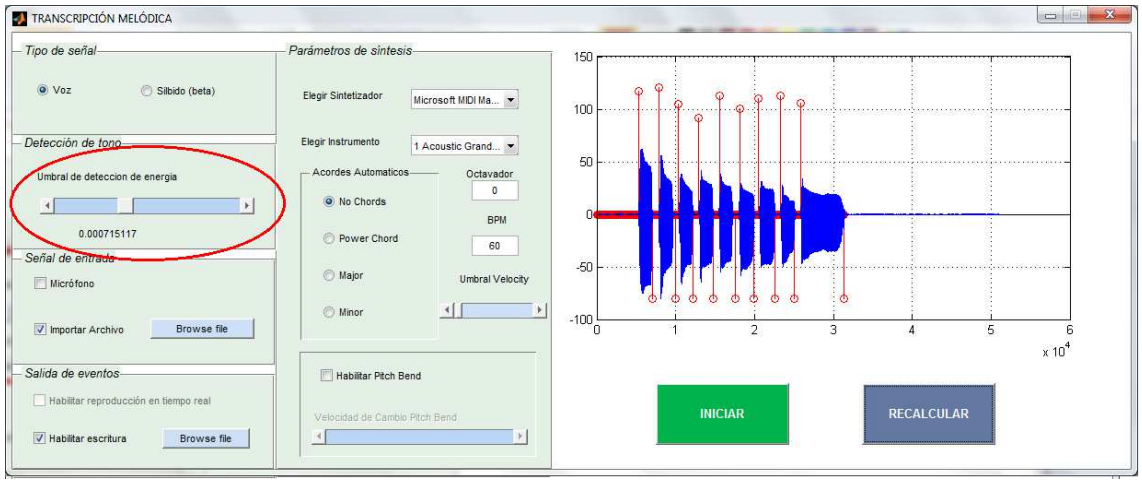

Figura 44: interfaz melódica resultante tras aumentar el umbral de detección, se puede observar como los errores han desaparecido.

- Elegir instrumento: Menú tipo *pop-up* con los diferentes instrumentos de los que se dispone. El sonido del instrumento variará evidentemente con la elección del sintetizador.
- Acordes automáticos: Se ha habilitado la posibilidad de crear acordes automáticos, tomando la nota producida por la voz como la fundamental. Los acordes son de triada (tres notas), y se incluyen tres posibilidades: mayor, menor, o de poder (*power chord*).

- Octavador: Ya que el registro de la voz no es tan amplio como para abarcar las bajas frecuencias de un bajo o el alto registro de un violín, se incluye la posibilidad de trasponer la interpretación las octavas que se deseen. De esta forma, un LA de 440Hz producido en la 4ª octava, si se selecciona '-1' en el octavador, sonara un LA de 220Hz, es decir, una octava por debajo.
- BPM: el fichero MIDI se graba con un tempo, este es designado por los *beats* por minuto (BPM). En este recuadro se informa de los BPM a los que se está grabando.
- Umbral Velocity: *slider* que permite variar la dinámica. Entendiendo como dinámica a la diferencia entre la nota con mayor volumen menos la nota con menor volumen (velocity). Al aumentar el valor del umbral de velocity, aumentamos la diferencia entre las notas más y menos sonoras.

Pitch Bend

Al habilitar la opción de *Pitch Bend* se habilita la variación de tono a lo largo de una misma nota. En los sintetizadores comunes, se suele utilizar un teclado para tocar. Un teclado no puede variar el tono de una misma nota como el violín o cualquier instrumento de cuerda sin trastes. La solución que se dio fue incluir la rueda de modulación. Moviendo esta rueda, se conseguía variar el tono.

Ahora bien, mediante la voz sí que se puede variar el tono de una nota. Esta variación tendrá el mismo resultado que si se estuviera accionando la rueda de modulación. Ahora bien, la rueda tiene ciertas limitaciones, es decir, esta tiene un valor máximo y mínimo, más allá del cual no se puede variar más la frecuencia de la nota. Esto implica que si el tono de la voz se va variando progresivamente durante una misma nota, llegara un momento en el que se dejara de apreciar cambio.

Iniciar/Detener

Este botón se habilita cuando se ha seleccionado la fuente. Al pulsarse, comienza la transcripción ya sea en tiempo real, u offline. Una vez pulsado, cambia su color y cadena de caracteres, siendo el botón de "DETENER", si éste es pulsado, finaliza la grabación.

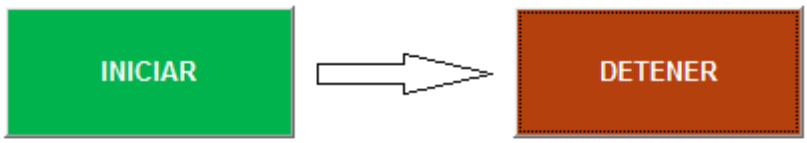

Figura 45:Evolución del botón INICIAR / DETENER, antes de comenzarse la transcripción (izquierda) y en medio de la grabación (derecha).

Recalcular

Si se da el caso de haber seleccionado el micrófono como fuente de audio, y se está grabando en un archivo, cada vez que se pulse en "INICIAR", se iniciará una nueva grabación. Pero si lo que deseamos es calcular otra vez la transcripción de una señal ya grabada (por haber modificado algún parámetro como acordes, umbral de detección etc.), se tendrá que pulsar en "RECALCULAR".

6.3 GUI: Transcripción Rítmica

En la GUI correspondiente a la transcripción rítmica, será donde el usuario configurará el sistema (grabación de golpes para entrenamiento, elección de red neuronal etc.) y donde podrá observar y modificar la transcripción.

Se pasa a detallar la utilidad de cada una de las secciones:

Grabar golpes

En esta fase se crea el set de datos necesario para entrenar la red neuronal.

- Grabar: El usuario debe en primer lugar grabar los golpes de cada una de las clases (pulsando los botones "GRABAR" rojos Figura 46). Una vez comenzada la grabación comienza la medición del umbral de ruido, el sistema muestra el mensaje de la Figura 47, con una barra de progreso.
- Una vez se ha medido el ruido ambiente se informa al usuario que debe golpear la superficie elegida repetidas veces Figura 48.
- Si el usuario a dado suficientes golpes para realizar un buen entrenamiento, se le informa que la entrada ha sido correctamente creada (Figura 49 a) y que debe validarla en el menú de grabación. En caso contrario, un mensaje de error (Figura 49 b) es reflejado.

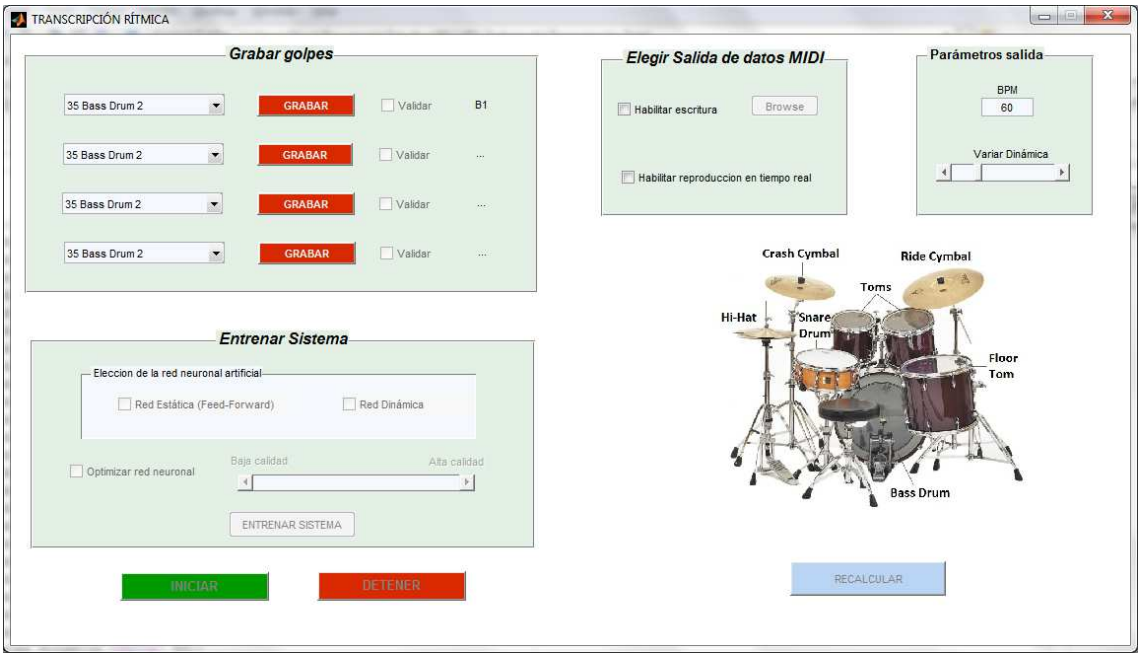

Figura 46: GUI de transcripción rítmica.

Figura 47: mensaje de grabación de ruido.

Figura 48: Información relevante a la creación de las entradas.

Figura 49: Mensajes que se muestra al crearse correctamente la entrada (a), y de error (b).

- Validar: como se ha comentado, cuando una entrada es grabada correctamente, y el usuario considera que es correcta, debe validarla.
- Instrumento: El menú *pop-up* que se encuentra al lado de los botones de grabar representan los instrumentos de percusión que se disponen. Hay que decir que no existe un instrumento llamado "batería", hay que componer la batería con los diferentes elementos: bombo (*bass drum*) caja (*snare drum*) charles (*hi-hat*), platillo (*crash cymbal*)… Ya que éstos están en ingles, se ha incluido un esquema de una batería indicando los nombres de los elementos para facilitar la elección. Existen también elementos de percusión latina (bongos, congas etc. etc.)
- Nota: En MIDI, cada instrumento de percusión está relacionado con una nota, es decir, el bombo se corresponde a un SI de la primera octava (B1 en notación americana) como se puede ver en la Figura 46. Cuando lo que se desea es importar el archivo MIDI a un secuenciador, cada elemento quedará reflejado como inicios y finales de una misma nota. Para facilitar el uso de este fichero en los secuenciadores, se informa de la nota a la que corresponde cada instrumento mediante el texto estático que se encuentra a la derecha de la caja de validación.

Entrenar Sistema

Una vez grabados y validados todos los golpes, se habilita la posibilidad de elegir la red a entrenar, y en el caso de la red estática, también de optimizarla.

El grado de optimización viene reflejado en la Tabla 4.

Habiendo elegido ya la red neuronal, se entrena la red pulsando en "Entrenar Sistema". Dependiendo de la red elegida y si se ha seleccionado optimización, el tiempo de entrenamiento será diferente.

Elegir salida de datos MIDI

Es necesario decidir si deseamos escuchar el sonido de los instrumentos en tiempo real, accionando "Habilitar reproducción en tiempo real", o si deseamos almacenar los datos en un fichero. Si elegimos esta segunda, se debe navegar por el sistema para guardar el archivo mediante el botón "Browse". Si no se especifica nombre, el sistema guarda un fichero con el nombre *eventos_trans_ritmica.txt*.

Parámetros de salida

En éste se especifican los valores de los *beats* por minuto con el mismo objetivo que se explico en la GUI melódica. Y la posibilidad de variar la dinámica, parámetro análogo a "umbral velocity" de la interfaz melódica.

Iniciar/Detener

Ya configurado el sistema, solo queda iniciar la transcripción accionando el botón "INICIAR". En el momento que se desee finalizarla, solo habrá que pulsar "DETENER".

Recalcular

En el caso en el que se haya seleccionado la grabación de los datos MIDI en un fichero. La imagen del esquema de la batería es sustituido por la transcripción realizada. Cada evento es representado por un delta, de valor igual a la velocity del mismo (Figura 50).

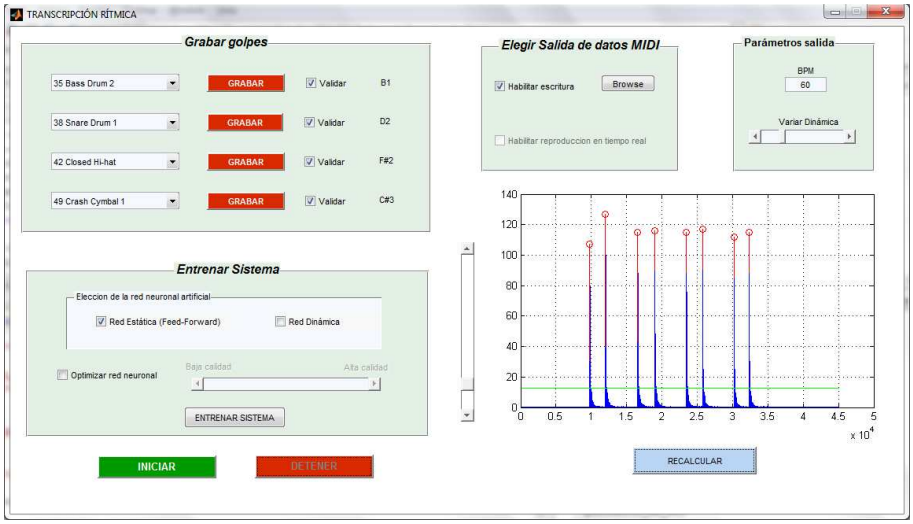

Figura 50: GUI de transcripción rítmica tras haber grabado un ritmo. Puede observarse el valor de velocity de cada golpe (rojo), la envolvente de la señal de audio (azul).

PARTE III

Resultados, conclusiones y líneas futuras de desarrollo.

Capítulo 7

Resutados y conclusiones

7.1 Transcripción rítmica

En esta sección se pasa a probar el sistema de transcripción de impactos sobre objetos a eventos MIDI.

Base de datos

La base de datos que va a poner a prueba el sistema se compone de los siguientes elementos usados como baquetas:

- nudillos de la mano
- lapicero
- cuchara

Que impactarán sobre:

- mesa de madera
- libreta de papel
- caja de madera
- caja de cartón

- lámpara metálica

Las combinaciones de elementos de excitación y excitados componen una base de datos total de 5 sets.

Aplicación

Al disponerse de dos redes neuronales, estos 5 sets se aplicaran por separado a ambas redes.

Se recuerda que en el caso de la red de propagación hacia adelante, existe la posibilidad de ser optimizada. En las siguientes pruebas realizadas no se utilizará esta mejora, haciéndose uso únicamente de la red por defecto.

Tras entrenar las redes, se van a realizar 20 impactos en cada uno de los objetos asociados a cada clase (80 golpes en total), y a medirse el numero de errores cometidos.

Tabla 6: Tasa de error del sistema de transcripción rítmica para cada set de la base de datos. Refiriéndose con *E* a la red estática, y *D* a la dinámica.

	SET 1		SET 2		SET 3		SET 4		SET 5	
	E	D	E	D	E	D	E	D	E	D
Clase 1	0	2	2	1	0	0	0	1	0	1
Clase 2	0	3	0	1	0	2	0	4	0	1
Clase 3	0	4	0	2	0	1	1	2	0	2
Clase 4	0	5	0	3	0	2	0	1	0	4
Tasa de error (%)	**0**	**17,5**	**2,5**	**8,75**	**0**	**6,25**	**1,25**	**10**	**0**	**10**

Tabla 7: Tasa de error total, y tiempo de procesado de la base de datos por cada red.

	Tasa de error total (%)	Tiempo de procesado (ms)
Estática	0,75	13,2
Dinámica	10,50	8,4

Como puede observarse en la Tabla 7, la tasa de error de la red estática es muy inferior a la dinámica alrededor de 14 veces menor. Esto es debido a la solida extracción de

características y su posterior selección. En la red dinámica, la señal de entrada es la correspondiente al tramo de ataque temporal, por lo que está sujeta a las pequeñas variaciones en el impacto de los elementos, y depende enormemente de la capacidad del usuario de golpear los objetos en el mismo sitio, con la misma parte del elemento de excitación etc.

Destacar los buenos resultados de la red *feed forward* teniendo en cuenta que se está utilizando únicamente la red por defecto.

De todas formas, y recordando el objetivo especifico de cada red, la estática tenía el objetivo de una clasificación robusta pero lenta, y la dinámica debía ser rápida.

Por ende, se ha de tener en cuenta el tiempo de respuesta de la red, siendo éste el total desde que se extraen las características, hasta que se envían los datos MIDI al sintetizador. Se interpretan un total de 80 golpes para cada red, tomando el valor medio del tiempo de procesado.

Los resultados que se muestran en la Tabla 7 demuestran que, en efecto, la red dinámica es más rápida, aunque con el precio de una mayor tasa de error. En cuanto al tiempo de procesado de la red estática, éste varia de un set a otro, ya que depende enormemente del número y tipo de características que haya sido seleccionado.

Hay que tener en cuenta que la red dinámica obtiene su información del ataque de la señal, es decir, la señal que procesa posee información relativa tanto al objeto que golpea, como el que es golpeado. En cambio, la red *feed forward* analiza mayormente aspectos relativos al espectro, mas propios del elemento golpeado.

Esto quiere decir que si el set elegido por el usuario comprende varios excitadores y mismas superficies de excitación, es muy posible que la red dinámica tenga mejores resultados. En cambio si se dispone de varios elementos que golpear, se sugiere utilizar la red *feed forward*.

Ejemplo real

Una vez observadas las capacidades del sistema, se va a probar mediante la interpretación de distintos ritmos, para observar su comportamiento.

Activando la red *feed forward* se golpean las diferentes superficies para conformar el ritmo reflejado en la Figura 51.

Figura 51: Representación del fichero MIDI creado en base a un ritmo. A la izquierda se muestran los diferentes elementos, y abajo la *velocity* de cada uno de ellos. Secuenciador *Ableton Live.*

El retardo que se obtiene al activar la función de tiempo real, no cumple las condiciones para ser integrado por el oído como "real". Este molesto retardo dificulta la interpretación a la vez que se escucha, pero no afecta al análisis *offline*.

El sistema pierde cierta capacidad de clasificación cuando los impactos se suceden muy seguidos. Esto es porque parte del golpe precedente entra en la ventana siguiente, entorpeciendo el segmentado.

7.2 Transcripción de voz

Se va a probar el algoritmo *offline* que convierte la voz en mensajes MIDI

Escalas

Las escalas evalúan de forma muy clara el funcionamiento del sistema, ya que el oído humano está muy entrenado en ellas. Se han interpretado dos escalas. En la primera de ellas el cantante interpreta una escala sin haber escuchado referencia musical alguna. En la segunda, la escala es cantada mientras se escucha la nota fundamental de la escala a través de unos auriculares.

Ambas escalas constan de 8 notas, empezando en un DO de la tercera octava, y finalizando en el DO de la cuarta.

Debido a la forma en la que se obtienen las notas, (midiendo los intervalos, y no la frecuencia con respecto a la referencia absoluta), en ambas transcripciones se produce el

mismo error. En el transcurso de la escala, la interpretación no es perfecta y se asigna la siguiente nota con un semitono de error, siendo arrastrado hasta el final de la interpretación.

En el caso de cantarse sin referencia, el error se produce en la segunda nota (un semitono por encima), y en la escala con referencia en el séptimo evento (también un semitono).

Este error que afecta a gran parte de la interpretación no supone ningún problema tratándose de MIDI, lo único que debe hacer el usuario es seleccionar el tramo desafinado (en realidad, afinado en exceso) y transponerlo un semitono.

Canción sencilla

Se tararea la conocida canción *cumpleaños feliz* como prueba de sencillez. La representación sobre un piano del fichero MIDI obtenido se muestra en la Figura 52 .

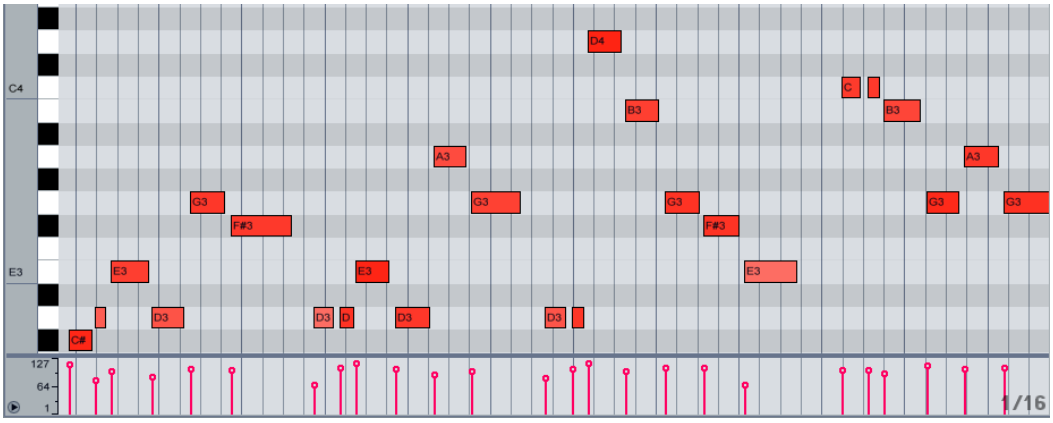

Figura 52: Representación en piano roll de la canción *cumpleaños feliz* donde existe un error en la primera nota.

En este caso, se ha cometido un error en la interpretación de la primera nota, la cual debería ser un RE de la 3ª octava. Los errores dependen en gran medida de la interpretación del cantante, pero se observa que para 24 notas cantadas, no ha habido ninguna detección errónea u nota omitida.

Canción de dos pistas

Como composición melódica de complejidad moderada se propone la línea principal de violines de la canción clásica de Tomaso Albinoni *Adagio in G Minor*, junto al

acompañamiento de violonchelos. Paliando los ya conocidos errores de referencia se puede conseguir el tramo principal de unos 30 segundos que se muestra en la Figura 53.

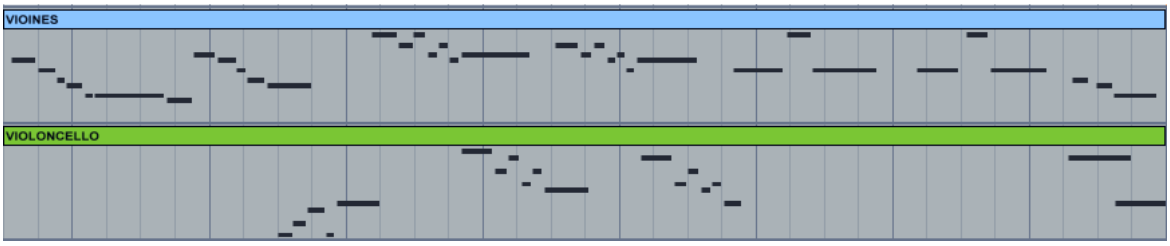

Figura 53: Fragmento de *Adagio in G Minor* de Tomaso Albinoni, resultante de la transcipción de voz.

Cabe destacar, que el sistema ha sido probado con otras fuentes musicales como lo es por ejemplo una guitarra acústica. El algoritmo es capaz de transcribir de forma monofónica la interpretación de una guitarra de forma fiable, dando algunos errores de octava por los fuertes armónicos de los instrumentos de cuerda.

7.3 Conclusiones

El objetivo que se perseguía con este proyecto era el de facilitar la creación artística de un usuario sin conocimiento de teoría ni practica musical.

Transcripción de voz

El sistema de composición de melodías logra traducir las notas tarareadas o cantadas eficientemente, pudiéndose componer canciones tanto sencillas como de complejidad moderada.

La forma de transcribir intervalos hace que no haya necesidad de afinar la voz, solo que la distancia en frecuencia de las notas sea la correcta. Dando el mismo resultado cuando el cantante interpreta con referencia que sin ella

Ha quedado demostrada la eficiencia del algoritmo de autocorrelación y sus variaciones, en el seguimiento del tono de la voz cantada. Destacar por otra parte la segmentación de eventos de eventos mediante el análisis de los acentos musicales basándose en la derivada relativa a la envolvente.

La función de tiempo real tiene una tiempo de respuesta corto dado el rápido procesado en el dominio del tiempo de la detección de tono, y la máquina de estados. Pero como ya se preveía, el conjunto de tiempo en la captación y procesado supera los límites en el tiempo de integración del oído humano.

Transcripción rítmica

El segundo sistema ofrece la posibilidad de componer ritmos de percusión golpeando diferentes objetos.

Se ha constatado la eficiencia de las redes neuronales como clasificador de patrones musicales. Dando excelentes resultados en una de las familias de señales musicales más complejas, los sonidos percusivos sin tono (*unpitched*).

De las dos redes neuronales diseñadas, la topología estática es la que ofrece mejores resultados, afianzándose la extracción y selección de características globales como un método eficiente de clasificación. Por otra parte, el tiempo invertido en esta tarea dificulta su utilización en tiempo real. A favor de la red dinámica (Elman), se observa que es más rápida, y por ende, más propensa a este tipo de aplicaciones.

El sistema implementado cumple las expectativas planteadas. El usuario puede componer una canción mediante su voz y algunos objetos. Si además, se procede a una fase de corrección de errores tras la transcripción, de gran sencillez dadas las facilidades del formato MIDI, el presente proyecto conforma una potente herramienta de composición, pudiendo ser utilizada con fines profesionales.

7.4 Líneas futuras de desarrollo

Como ya se sabía de antemano, la implementación en lenguaje de alto nivel hacen que la toma de datos junto con el procesado no cumplan los requisitos de velocidad para que pueda considerarse tiempo real. En este aspecto, la función de transcripción de voz es la que más cerca está de cumplir el objetivo. Por lo que un paso a dar sería la implementación de este sistema en un lenguaje de bajo nivel como pueda ser java o *C++*.

A lo hora de crear una composición, es necesario escuchar las pistas previamente grabadas a la vez que se van creando nuevas líneas (el llamado *foldback*), consiguiendo que el

conjunto esté sincronizado. La mayoría de aplicaciones con estas características toma la forma de instrumento virtual VST, pasado a ser un modulo de la estación de audio digital (*DAW*) Sería imprescindible en cuanto comodidad y funcionalidad, la implementación del sistema en este tipo de formato.

En cuanto a la transcripción rítmica, está la limitación de la monofonía, es decir, el sistema está preparado para analizar tan solo un golpe en cada instante. Y dado que algo muy común en las interpretaciones de batería es que se toquen por ejemplo bombo y crash (platillo) simultáneamente, sería imprescindible ofrecer esta posibilidad. Para ello se debería utilizar técnicas de discriminación de fuente sonora.

Sería interesante incluir mas clases en la transcripción, para poder crear bases rítmicas más ricas de elementos. Esto aumentaría la tasa de error del sistema por lo que habría que crear un sistema de detección más robusto para matener sus resultados. Una forma sería el combinar los resultados las redes de topología estática y dinámica. Una de ellas tendria en cuenta el patron temporal de la señal, y la otra extraería las caracteristicas temporales. Cabe decir que esa idea se probó con las dos redes implementadas, pero el conjunto de ambas obtenía peores resultados que la red *feedforward* actuando individualmente.

El error más frecuente en el sistema de transcripción melódica es la excesiva afinación que resulta al evaluar los intervalos entre notas, sin referencia alguna. Esto es útil para cantantes inexpertos. Pero para un usuario entrenado en el canto sería interesante darle la posibilidad de definir el tipo de referencia.

Este proyecto se ha basado en la transcripción de señales musicales, como son la voz o patrones rítmicos, en líneas musicales de mayor belleza acústica (haciendo uso del MIDI). Pero, ¿y si en vez de convertir música en música, se pudiera hacer uso de otro tipo de señales de entrada para la transcripción?. El cerebro humano puede realizar este tipo de "conversiones". Así como un compositor de bandas sonoras imagina la música que acompaña las imágenes de la película, una red neuronal podría aprender a extraer líneas musicales de una señal de video. O a transcribir las palabras de un poema. Las posibilidades de las redes neuronales son infinitas, casi tantas como las de un cerebro humano… ¿o tal vez mas?.

ANEXOS

ANEXO I: Diagramas de flujo.

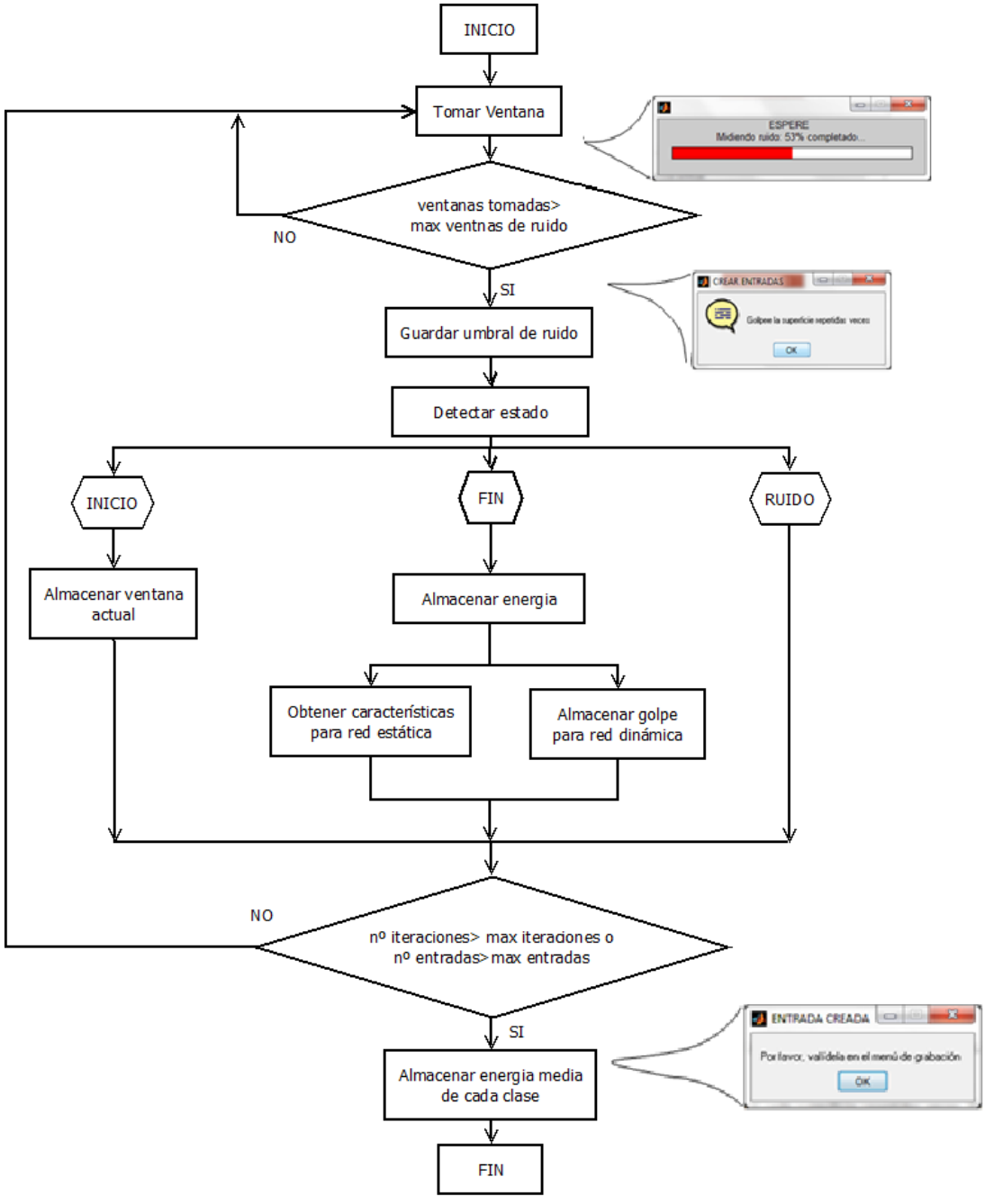

Figura 54: Sistema de transcripción rítmica. Diagrama de flujo correspondiente a la grabación del set de entrenamiento.

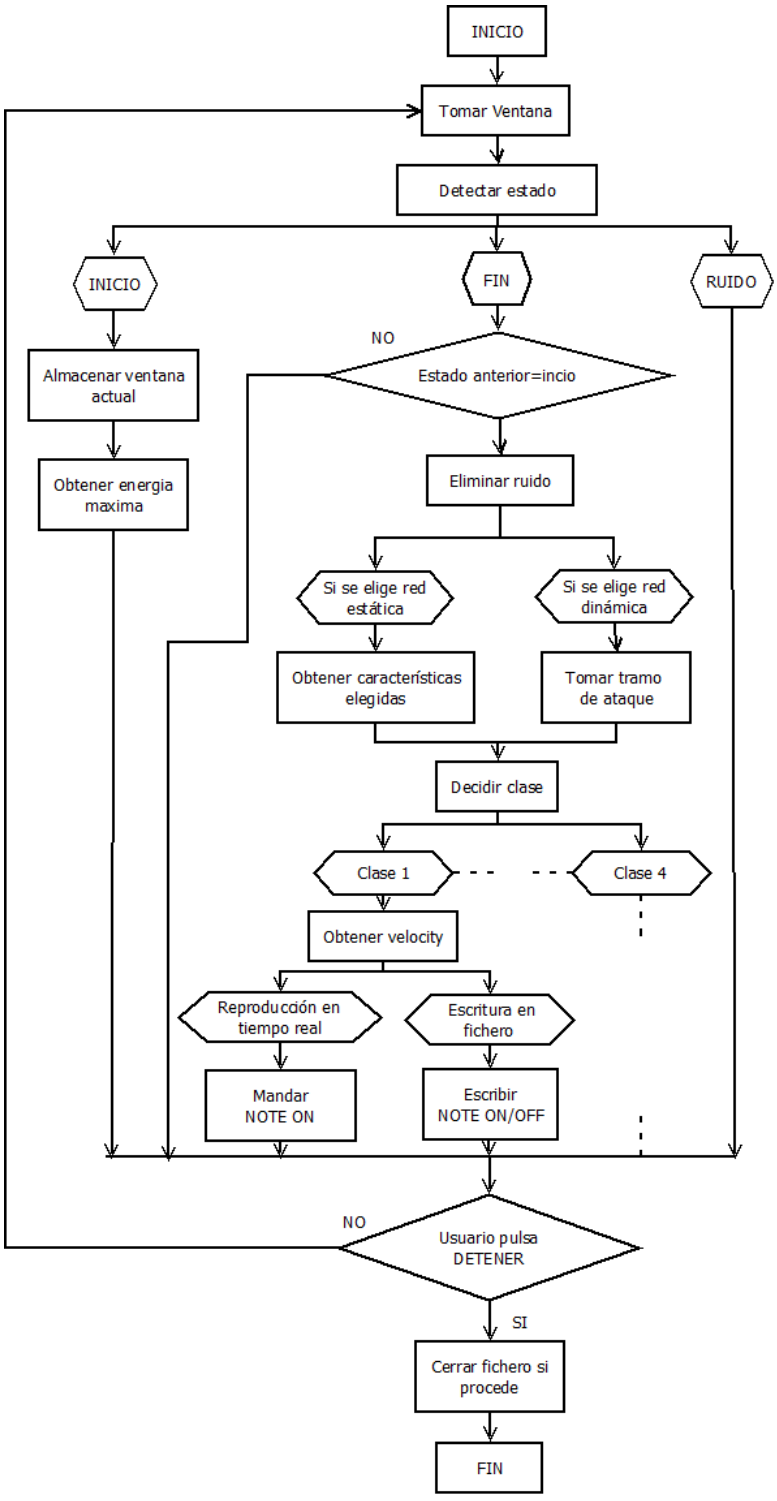

Figura 55: Diagrama correspondiente al funcionamiento del sistema de transcripción rítmica en fase de identificación.

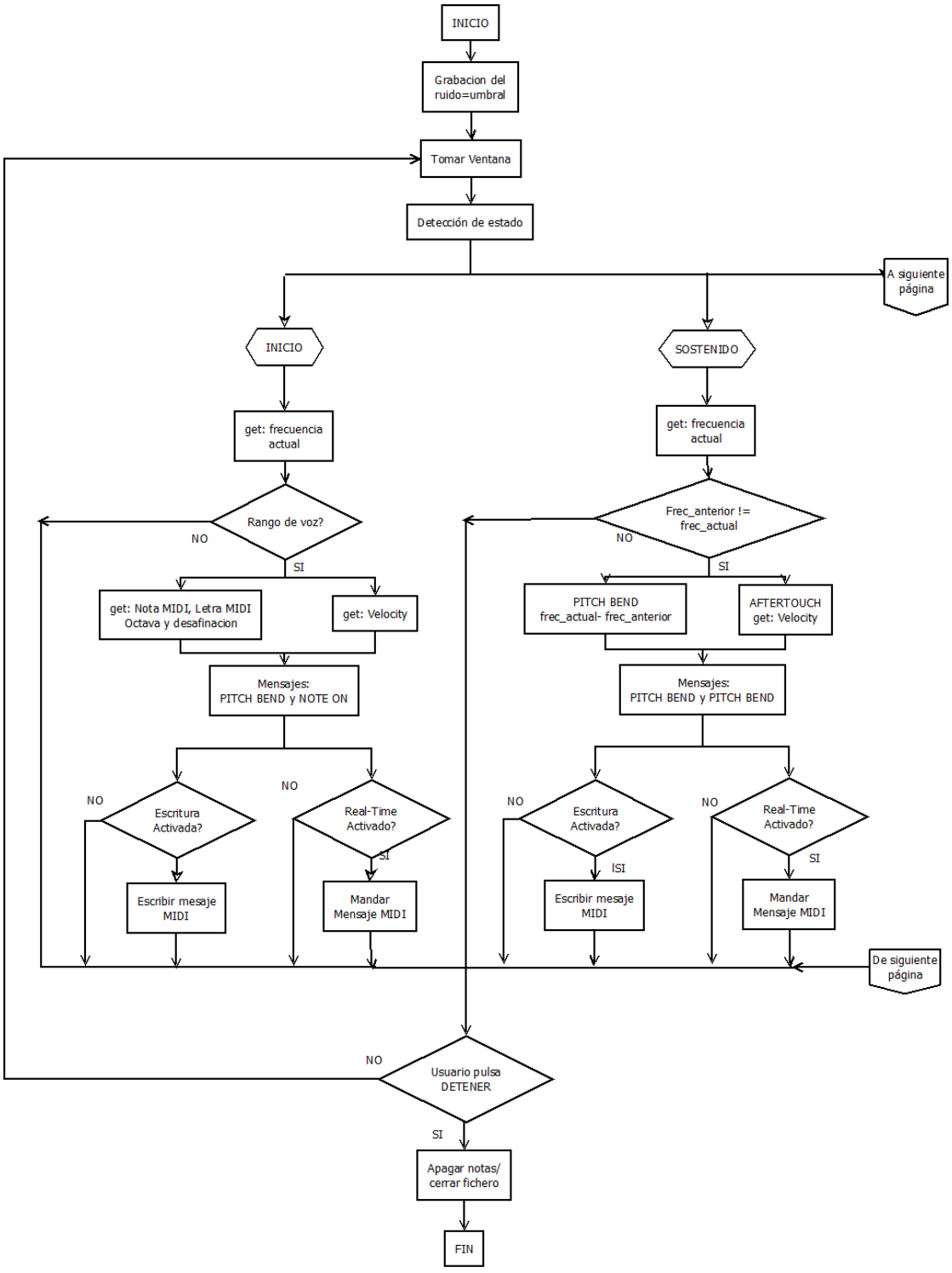

Figura 56: Ordinograma correspondiente al sistema de transcripción de voz. Parte 1.

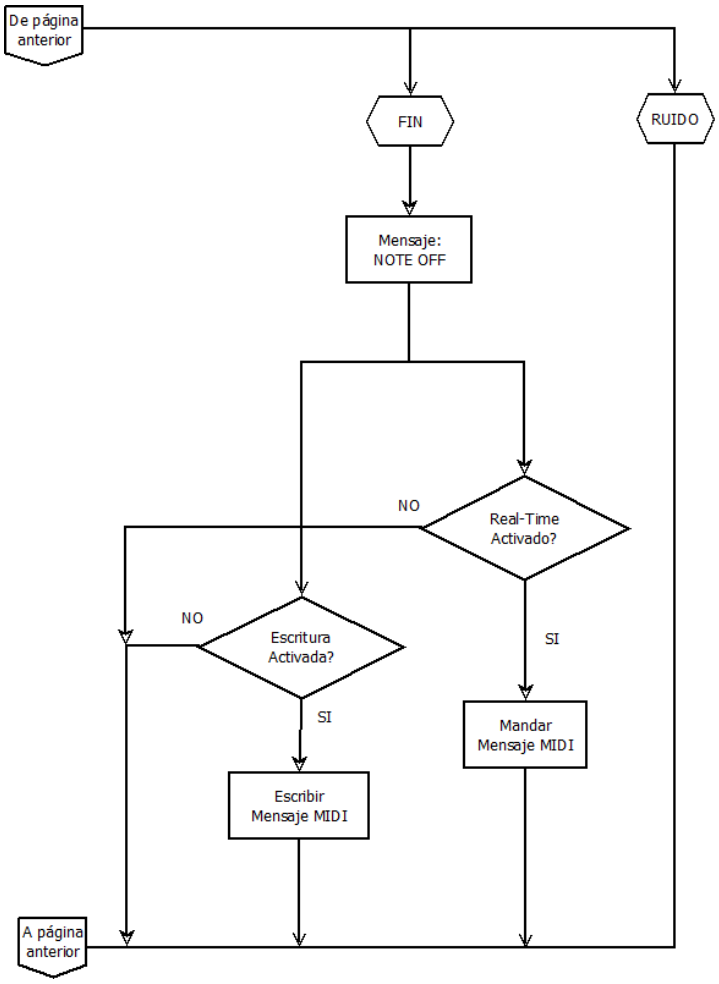

Figura 57: Ordinograma correspondiente al sistema de transcripción de voz. Parte 2.

ANEXO II: MIDI

Ejemplo de fichero MIDI texto. Tres notas (D3, G3 y A#3) con modulación de tono.

```
MThd | # of Tracks=1 | Division=800

 Track #1
       | Tempo | BPM= 90
   1:  1:491 |On Note    | chan=1 |  pitch= D 3  3 | vol=    94
   1:  1:491 |Pitch Wheel | chan=1 |  bend= 2176
   1:  1:531 |Pitch Wheel | chan=1 |  bend= 2176
   1:  1:551 |Pitch Wheel | chan=1 |  bend= 2560
   1:  1:571 |Pitch Wheel | chan=1 |  bend= 2560
   1:  1:591 |Pitch Wheel | chan=1 |  bend= 2944
   1:  1:611 |Pitch Wheel | chan=1 |  bend= 2944
   1:  1:881 |Off Note   | chan=1 |  pitch= D 3  3 | vol=    94
   1:  1:881 |Pitch Wheel | chan=1 |  bend=   0
   1:  2:088 |On Note    | chan=1 |  pitch= G 3  3 | vol=   127
   1:  2:088 |Pitch Wheel | chan=1 |  bend= 2304
   1:  2:128 |Pitch Wheel | chan=1 |  bend= 2304
   1:  2:148 |Pitch Wheel | chan=1 |  bend= 3200
   1:  2:168 |Pitch Wheel | chan=1 |  bend= 3200
   1:  2:188 |Pitch Wheel | chan=1 |  bend= 2560
   1:  2:452 |Off Note   | chan=1 |  pitch= G 3  3 | vol=   127
   1:  2:452 |Pitch Wheel | chan=1 |  bend=   0
   1:  2:621 |On Note    | chan=1 |  pitch= A#3  3 | vol=   105
   1:  2:621 |Pitch Wheel | chan=1 |  bend= 2304
   1:  2:661 |Pitch Wheel | chan=1 |  bend= 2304
   1:  2:681 |Pitch Wheel | chan=1 |  bend= 2304
   1:  2:701 |Pitch Wheel | chan=1 |  bend= 2304
   1:  2:721 |Pitch Wheel | chan=1 |  bend= 3072
   1:  2:741 |Pitch Wheel | chan=1 |  bend= 3072
   1:  2:761 |Pitch Wheel | chan=1 |  bend= 3072
   1:  2:781 |Pitch Wheel | chan=1 |  bend= 3072
   1:  2:801 |Pitch Wheel | chan=1 |  bend= 3072
   1:  2:821 |Pitch Wheel | chan=1 |  bend= 3072
   1:  2:841 |Pitch Wheel | chan=1 |  bend= 3072
   1:  2:861 |Pitch Wheel | chan=1 |  bend= 3072
   1:  2:881 |Pitch Wheel | chan=1 |  bend= 3968
   1:  2:901 |Pitch Wheel | chan=1 |  bend= 4864
   1:  2:921 |Pitch Wheel | chan=1 |  bend= 6016
   1:  2:941 |Pitch Wheel | chan=1 |  bend= 6016
   1:  2:961 |Pitch Wheel | chan=1 |  bend= 6016
   1:  2:981 |Pitch Wheel | chan=1 |  bend= 6016
   1:  3:001 |Pitch Wheel | chan=1 |  bend= 6016
   1:  3:021 |Pitch Wheel | chan=1 |  bend= 5120
   1:  3:495 |Off Note   | chan=1 |  pitch= A#3  3 | vol=   105
   1:  3:495 |Pitch Wheel | chan=1 |  bend=   0
       | End of track |
```

Tabla 8: Tabla de mensajes de voz. Aquellos que intervienen en la interpretación.

Table 1: MIDI 1.0 Specification Message Summary		
Status D7----D0	**Data Byte(s) D7----D0**	**Description**
Channel Voice Messages [nnnn = 0-15 (MIDI Channel Number 1-16)]		
1000nnnn	0kkkkkkk 0vvvvvvv	Note Off event. This message is sent when a note is released (ended). (kkkkkkk) is the key (note) number. (vvvvvvv) is the velocity.
1001nnnn	0kkkkkkk 0vvvvvvv	Note On event. This message is sent when a note is depressed (start). (kkkkkkk) is the key (note) number. (vvvvvvv) is the velocity.
1010nnnn	0kkkkkkk 0vvvvvvv	Polyphonic Key Pressure (Aftertouch). This message is most often sent by pressing down on the key after it "bottoms out". (kkkkkkk) is the key (note) number. (vvvvvvv) is the pressure value.
1011nnnn	0ccccccc 0vvvvvvv	Control Change. This message is sent when a controller value changes. Controllers include devices such as pedals and levers. Controller numbers 120-127 are reserved as "Channel Mode Messages" (below). (ccccccc) is the controller number (0-119). (vvvvvvv) is the controller value (0-127).
1100nnnn	0ppppppp	Program Change. This message sent when the patch number changes. (ppppppp) is the new program number.
1101nnnn	0vvvvvvv	Channel Pressure (After-touch). This message is most often sent by pressing down on the key after it "bottoms out". This message is different from polyphonic after-touch. Use this message to send the single greatest pressure value (of all the current depressed keys). (vvvvvvv) is the pressure value.
1110nnnn	0lllllll 0mmmmmmm	Pitch Wheel Change. 0mmmmmmm This message is sent to indicate a change in the pitch wheel. The pitch wheel is measured by a fourteen bit value. Center (no pitch change) is 2000H. Sensitivity is a function of the transmitter. (lllllll) are the least significant 7 bits. (mmmmmm) are the most significant 7 bits.

Tabla 9: Tabla de mensajes de modo y sistema.

Channel Mode Messages (See also Control Change, above)		
1011nnnn	0ccccccc 0vvvvvvv	Channel Mode Messages. This the same code as the Control Change (above), but implements Mode control and special message by using reserved controller numbers 120-127. The commands are:
		All Sound Off. When All Sound Off is received all oscillators will turn off, and their volume envelopes are set to zero as soon as possible. c = 120, v = 0: All Sound Off
		Reset All Controllers. When Reset All Controllers is received, all controller values are reset to their default values. (See specific Recommended Practices for defaults). c = 121, v = x: Value must only be zero unless otherwise allowed in a specific Recommended Practice.
		Local Control. When Local Control is Off, all devices on a given channel will respond only to data received over MIDI. Played data, etc. will be ignored. Local Control On restores the functions of the normal controllers. c = 122, v = 0: Local Control Off c = 122, v = 127: Local Control On
		All Notes Off. When an All Notes Off is received, all oscillators will turn off. c = 123, v = 0: All Notes Off (See text for description of actual mode commands.) c = 124, v = 0: Omni Mode Off c = 125, v = 0: Omni Mode On c = 126, v = M: Mono Mode On (Poly Off) where M is the number of channels (Omni Off) or 0 (Omni On) c = 127, v = 0: Poly Mode On (Mono Off) (Note: These four messages also cause All Notes Off)
System Common Messages		
11110000	0iiiiiii [0iiiiiii 0iiiiiii] 0ddddddd --- --- 0ddddddd 11110111	System Exclusive. This message type allows manufacturers to create their own messages (such as bulk dumps, patch parameters, and other non-spec data) and provides a mechanism for creating additional MIDI Specification messages. The Manufacturer's ID code (assigned by MMA or AMEI) is either 1 byte (0iiiiiii) or 3 bytes (0iiiiiii 0iiiiiii 0iiiiiii). Two of the 1 Byte IDs are reserved for extensions called Universal Exclusive Messages, which are not manufacturer-specific. If a device recognizes the ID code as its own (or as a supported Universal message) it will listen to the rest of the message (0ddddddd). Otherwise, the message will be ignored. (Note: Real-Time messages ONLY may be interleaved with a System Exclusive.)
11110001	0nnndddd	MIDI Time Code Quarter Frame. nnn = Message Type dddd = Values
11110010	0lllllll 0mmmmmmm	Song Position Pointer. This is an internal 14 bit register that holds the number of MIDI beats (1 beat= six MIDI clocks) since the start of the song. l is the LSB, m the MSB.
11110011	0sssssss	Song Select. The Song Select specifies which sequence or song is to be played.

REFERENCIAS

[1] J. A. Moorer. "On the segmentation and analysis of continuous musical sound by digital computer". PhD thesis, Department of Music, Stanford University, Stanford, CA, May 1975.

[2] I. Katayose. "The Kansei music system". *Computer Music Journal*, 1989.

[3] M. Hawley. "Structure out of Sound". PhD thesis, MIT Media Laboratory, 1993.

[4] T. Kashino. "A sound source separation system with the ability of automatic tone modeling". *Proceedings of the International Computer Music Conference*, 1993.

[5] W. A. Schloss. "On the automatic transcription of percussive music from acoustic signal to high-level analysis". PhD thesis, Center for Computer Research in Music and Acoustics, Stanford University, Stanford, California, USA, May 1985.

[6] M. Goto and Y. Muraoka."A beat tracking system for acoustic signals of music". In *Proc. ACM Multimedia*, 1994.

[7] A. Klapuri. "Sound onset detection by applying psychoacoustic knowledge". In *Proc. IEEE International Conference on Acoustics, Speech, and Signal* Processing, Phoenix, Arizona, USA, 1999.

[8] P. Herrera, A. Yeterian, and F. Gouyon. "Automatic classification of drum sounds: a comparison of feature selection methods and classification techniques". In *Proc. 2nd International Conference on Music and Artificial Intelligence*, Edinburgh, Scotland, UK, Sept. 2002.

[9] J.I. Shonle and K.E. Horan. "The pitch of vibrato tones". *Journal of the Acoustical Society of America,* January, 1980.

[10] L.P. Clarisse, J.P. Martens, M. Lesaffre, B. De Baets, H. De Meyer and M. Leman. "An auditory model based transcriber of singing sequences". In *International Conference on Music Information Retrieval,* Paris, France, October 2002.

[11] G. Haus and E. Pollastri. "An audio front end for query-by-humming systems". In *2nd Annual International Symposium on Music Information Retrieval,* Bloomington, Indiana, USA, 2001.

[12] R. J. McNab, L.A. Smith, and I.H. Witten. "Signal processing for melody transcription". In *19th Australasian Computer Science Conference,* Melbourne, Australia, February 1996.

[13] S. V. Vaseghi. *Advanced Digital Signal Processing and Noise Reduction.* John Wiley & Sons Ltd, West Sussex, 2006.

[14] T. Tolonen and M. Karjalainen. "A computationally efficient multipitch analysis model". In *IEEE Transactions on Speech and Audio Processing*, 2000.

[15] L. R. Rabiner and R. W. Schafer, *Digital Processing of Speech Signals,* Prentice Hall, 1978.

[16] A. Ghias, J. Logan, D. Chamberhn, and B.C. Smith. "Query by humming: Musical information retrieval in an audio database". In *ACM Multimedia Conference*, San Fransisco, California, November 1995.

[17] H. Shih, S.S. Narayanan, and C.-C.J. Kuo. "Multidimensional humming transcription using a statistical approach for query by humming systems". In *IEEE International Conference on Acoustics, Speech, and Signal Processing,* , Hong Kong, China, 2003.

[18] A. de Cheveigne and H. Kawahara. "YIN, a fundamental frequency estimator for speech and music". In *Journal of the Acoustical Society of America*, Abril, 2002.

[19] T. Viitaniemi, A. Klapuri, and A. Eronen. "A probabilistic model for the transcription of single-voice melodies". In *2003 Finnish Signal Processing Symposium,* Tampere, Finland, May 2003.

[20] A. Klapuri, A Eronen, and J. Astola. "Analysis of the meter of acoustic musical signals".In *IEEE Transactions on Speech and Audio Processing,* 2006.

[21] P. Herrera and J. Bonada. "Vibrato extraction and parametrization in the spectral modehng synthesis framework". In *First C0ST-G6 Workshop on Digital Audio Effects,* Barcelona, España, Noviembre 1998.

[22] J.P. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies, and M.B. Sandler. "A tutorial on onset detection in music signals". In *IEEE Transactions on Speech and Audio Processing* 2005.

[23] N. Collins. "A comparison of sound onset detection algorithms with emphasis on psychoacoustically motivated detection functions". In *Audio Engineering Society 118th Convention,* Barcelona, Spain, May 2005.

[24] A. Klapuri, M. Davy. *Signal processing, methods for music transcription*. Springer, 2000.

[25] F. Gouyon, P. Herrera, and P. Cano. "Pulse-dependent analysis of percussive music". In *22nd International Audio Engineering Society Conference,* Espoo, Finland, 2002.

[26] F. Gouyon, F. Pachet, and O. Delerue. "On the use of zero-crossing rate for an application of classification of percussive sounds". In *International Conferenceon Digital Audio Effects,* Verona, Italy, December 2000.

[27] O. Gillet and G. Richard. "Automatic transcription of drum loops". In *IEEE International Conference on Acoustics, Speech, and Signal Processing,* Montreal, Canada, May 2004.

[28] International Organization for Standardization. *ISO/IEC 15938-4:2002 Information Technology - Multimedia Content Description Interface - Part 4- Audio.* International Organization for Standardization, Geneva, Switzerland, 2002.

[29] D. Nguyen and B. Widrow, "The truck backer-upper: An example of self-learning in neural networks", In *Proc. Intl. Joint Conference on Neural Netwoks*, Washington DC, USA, 1989.

[30] T.J. Sejnowski and C.R.Rosenberg, "Nettalk: a parallel network that learns to read aloud", Tech.Rep. Johns Hopkins University, Baltimore, USA, 1986.

[31] P.M. Shea and V.Lin, "Detection of explosives in checked airline baggage using artificial neural system", In *Proc. Intl. Joint Conference on Neural Networks*, Washington DC, USA, 1989.

[32] D. Hebb. *The Organitation of Behavior*, John Wiley and songs, New York, 1949.

[33] H. Demuth and M., Beale. *Neural Network Toolbox-for Use with MATLAB User's Guide,* The Mathworks, Massachusetts, 2002

[34] L.R. Medsker and L.C. Jain. *Recurrent neural networks: design and applications*, Boca Raton, 2000.

[35] J.L Elman. *Finding structure in time*, Cognitive Science, 1990.

[36] D. Rumelhart, J. McClelland, and the PDP Research Group, "Parallel Distributed Processing: Explorations in the Microstructure of Cognition", MIT Press. 1986.

[37] G. Tesauro, "Neurogammon Wins Computer Olympiad", *Neural Computation*, 1989.

[38] M. J. D. Powell, "Restart procedures for the conjugate gradient method", Mathematical Programming, 1977.

[39] E.M.L. Beale, "A Derivation of Conjugate Gradients", in *Numerical Methods for Nonlinear Optimization*, F.A. Lootsma, Academic Press, London, 1972.

[40] O. Singh and N. Singla, "Optimization of Feedforward Neural Network for Audio Classification Systems", In *International Journal of advanced engineering sciences and technologies*, 2011.

[41] L. Badri, "Development of Neural Networks for Noise Reduction", Faculty of Engineering, Philadelphia University, Jordan. 2009.

[42] K. Kira and L. A. Rendell, "A practical approach to feature selection", In *Proc. 9th Int. Conf. Machine Learning*, Aberdeen, 1992.

[43] T. G Dietterich, "Machine learning research: Four current directions", In *AI Magazine*, *1997*.

ANEXO DE CODIGO

Codificador de voz MIDI

Oliver Vivar

Menú principal

```
function varargout = Menu_principal(varargin)
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Menu_principal_OpeningFcn, ...
                   'gui_OutputFcn',  @Menu_principal_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

% --- Executes just before Menu_principal is made visible.
function Menu_principal_OpeningFcn(hObject, eventdata, handles,
varargin)
imshow('imagen_main.jpg');
set(hObject,'name','MENU PRINCIPAL');
% Choose default command line output for Menu_principal
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);


% --- Outputs from this function are returned to the command line.
function varargout = Menu_principal_OutputFcn(hObject, eventdata,
handles)
varargout{1} = handles.output;


% --- Executes on button press in melodica.
function melodica_Callback(hObject, eventdata, handles)
GUI_funcionMelodica;


% --- Executes on button press in ritmica.
function ritmica_Callback(hObject, eventdata, handles)
percusion_nn_GUI;


% --- Executes during object creation, after setting all properties.
function dibujo_CreateFcn(hObject, eventdata, handles)

function edit1_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end
```

Transcripcion melódica

```
function varargout = GUI_funcionMelodica(varargin)
%
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @GUI_funcionMelodica_OpeningFcn, ...
    'gui_OutputFcn',  @GUI_funcionMelodica_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before GUI_funcionMelodica is made visible.
function GUI_funcionMelodica_OpeningFcn(hObject, eventdata, handles,
varargin)

% Choose default command line output for GUI_funcionMelodica
handles.output = hObject;
set(hObject,'name','TRANSCRIPCIÓN MELÓDICA');
% Update handles structure
guidata(hObject, handles);
%abrir la entrada targeta de sonido e inicializarla
handles.ai=analoginput('winsound');
addchannel(handles.ai,1);
fs=8000;
ventana=240;
step=ventana/2;

set(handles.ai,...
    'SampleRate',fs,...
    'SamplesPerTrigger',step,...
    'SamplesAcquiredFcnCount', step,...
    'TriggerRepeat', inf,...    %numero de iteraciones
    'StopFcn', @apagar_notas, ...
    'SamplesAcquiredFcn', @analisis_reproduccion, ...
    'Tag', 'ai_tag'...
    );

%Obtención de dispositivos MIDI disponibles
import javax.sound.midi.*;
dispositivos = MidiSystem.getMidiDeviceInfo;
for i = 1:length(dispositivos)
    nombres_disp(i) = dispositivos(i).getName;
```

```
end

imshow('Icono Funcion melodica.jpg');

set(handles.Sintetizadores,'String',char(nombres_disp));

%inicializar botones
set(handles.Escribir,'Value',0)
set(handles.real_time,'Value',0,...
    'Enable','on');
set(handles.Octavar,'String',0);
set(handles.vel_cambio_pitch_bend,'Enable','off');
set(handles.text3,'Enable','off');
set(handles.habilitar_bend,'Value',0);
set(handles.busca_fichero,'Enable','off');
set(handles.voz,'Value',1);
set(handles.Iniciar,'String','INICIAR',...
    'BackgroundColor',[0 0.7 0.3],...
    'Enable','off');
set(handles.recalcular,'Enable','off');
set(handles.velocity,'Value',0.01);
set(handles.microfono,'Value',0);
set(handles.importar,'Value',0);
set(handles.importar_browse,'Enable','off');
set(handles.Umbral_energia,'Value',2);
set(handles.Octavar,'String','0');


%iniciar valores
assignin('base','disp_midi_elegido',1);
assignin('base','instrumento',0);
assignin('base','umb_energia',9e-4);
assignin('base','vel_bend',4);
assignin('base','octava',0);
assignin('base','acorde','no_chord');
assignin('base','escritura_habilitada',0);
assignin('base','real_time',0);
assignin('base','pb_on',0);
assignin('base','voz',1);
assignin('base','silbido',0);
assignin('base','control_inicio',1);
assignin('base','minimo',0.01);
assignin('base','fichero_path_nombre','eventos.txt');
assignin('base','microfono',0);
assignin('base','importar',0);
assignin('base','bpm','60');

%medimos el ruido ambiente:
fs = 8000;
h = msgbox('Grabando ruido ambiente. Espere por
favor...','CARGANDO','help') ;
ruido = (max(wavrecord(3*fs, fs)))^2; %3 es el numero de segundos para
la medicion.
close(h);

assignin('base','umb_energia',ruido);
assignin('base','umb_energia_auto',ruido);
set(handles.valor_umbral_deteccion,'String',ruido);
guidata(hObject, handles);
```

```
% UIWAIT makes GUI_funcionMelodica wait for user response (see
UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = GUI_funcionMelodica_OutputFcn(hObject, eventdata,
handles)
% Get default command line output from handles structure
varargout{1} = handles.output;



% --- Executes on button press in Iniciar.
function Iniciar_Callback(hObject, eventdata, handles)
escribir=evalin('base','escritura_habilitada');
real_time=evalin('base','real_time');
microfono=evalin('base','microfono');
importar=evalin('base','importar');
umb_energia=evalin('base','umb_energia');
vel_bend=evalin('base','vel_bend');
instrumento=evalin('base','instrumento');
octava=evalin('base','octava');
disp_midi=evalin('base','disp_midi_elegido');
acorde=evalin('base','acorde');
pb_on=evalin('base','pb_on');
control_inicio=evalin('base','control_inicio');
BPM=evalin('base','bpm');

if ~real_time && ~escribir
    msgbox('Por favor, seleccione reproduccion y/o escritura en
fichero','AYUDA');
else

    %señal de entrada elegida
    voz=evalin('base','voz');
    if escribir
        nombre_fichero=evalin('base','fichero_path_nombre');
    else
        nombre_fichero='no escribir';
    end

    if voz==1
        signal='voz';
    else
        signal='silbido';
    end

    if real_time && control_inicio
        set (handles.Iniciar,'String','DETENER');
        set (handles.Iniciar,'BAckgroundColor',[0.7 0.25 0.05]);

        %parametros captacion de audio.
        ventana=240;
        s.sig=ventana/2;
        s.fs=8000;
        s.ventana_anterior = zeros(s.sig,1);
        %parametros de detección, y decision de estado
        s.energ_min=0.01;
        s.umb_derivada=0.08;
        s.umb_energ=umb_energia;
```

```
        s.energ_max=0;
        s.nota_ant=0;
        s.f_act_vent=0;
        s.f_ant_vent=0;
        s.estado_anterior='ruido';
        s.FI=6; %factor de interpolacion
        %parametros de salida de datos MIDI
        s.contador=2;
        s.channel=0;
        s.octavar=octava+1;
        s.nota=0;
        s.bend=64;
        s.pb_on=pb_on;
        s.acordes=acorde; %tipo de acorde elgido
        s.acorde=cell(3,2); %notas del acorde
        s.vel_bend=vel_bend;
        s.programa=instrumento;
        s.escribir=escribir;
        s.nombre=nombre_fichero;
        s.real_time=real_time;
        %para modo offline
        s.temporal=[]; %almacena la señal completa.

        import javax.sound.midi.*;
        s.devices = MidiSystem.getMidiDeviceInfo;
        s.mens = ShortMessage;

        s.midiout = MidiSystem.getMidiDevice(s.devices(disp_midi));
%Seleccion disp MIDI
        s.midiout.open;

        s.midi.recep = s.midiout.getReceiver;


        %programa elegido:
        s.mens.setMessage(ShortMessage.PROGRAM_CHANGE,instrumento,
s.channel);
        s.midi.recep.send(s.mens, -1);

        %abrimos el fichero de escritura midi si procede
        if s.escribir==1
            s.file = fopen(s.nombre,'w');
            fprintf(s.file,['MThd | # of Tracks=1 | '...
                'Division=1000 \n\n Track #1 \n       | Tempo | BPM=
%s \n'],BPM);
        end

        %modificar ancho de banda dependiendo de la fuente
        s.signal=signal;
        if strcmp(signal,'voz')
            s.rango=[82,1047];
        else
            s.rango=[500,5000];
        end
        %actualizar estructura
        set (handles.ai,'UserData',s);
        start(handles.ai);
        control_inicio=0;
        assignin('base','control_inicio',control_inicio);
```

```
    else if ~control_inicio
            stop(handles.ai);
            waittilstop(handles.ai,1);
            set (handles.Iniciar,'BAckgroundColor',[0 0.7 0.3]);
            set (handles.Iniciar,'String','INICIAR');
            assignin('base','control_inicio',1);
        end

    end

    if control_inicio && escribir
        if ~importar
            set (handles.Iniciar,'String','DETENER');
            set (handles.Iniciar,'BAckgroundColor',[0.7 0.25 0.05]);
            assignin('base','control_inicio',0);
        end
        if microfono
            r = audiorecorder(8000, 16, 1);
            set(r,'Tag','recording');
            record(r)
            handle_r=guidata(hObject);
            handle_r.r=r;
            guidata(hObject, handle_r);
        end


    else if ~control_inicio && escribir && ~real_time
            set (handles.Iniciar,'BAckgroundColor',[0 0.7 0.3]);
            set (handles.Iniciar,'String','INICIAR');
            if ~importar
                assignin('base','control_inicio',1);
            else
                assignin('base','control_inicio',0);
            end


            if escribir
                %adquirir datos de base
                umb_ruido=evalin('base','umb_energia');
                octavar=evalin('base','octava');
                máximo=1;
                minimo=evalin('base','minimo');
                %Abrir fichero de escritura

                file = fopen(nombre_fichero,'w');
                fprintf(file,['MThd | # of Tracks=1 | '...
                    'Division=1000 \n\n Track #1 \n     | Tempo | BPM=
%s \n'],BPM);



                %Obtener datos de audio
                if microfono && ~control_inicio
                    stop(handles.r)
                    grabacion=getaudiodata(handles.r);
                    assignin('base','grabacion',grabacion);
                    fs=get(handles.r,'SampleRate');
                else if importar
                        grabacion=evalin('base','grabacion');
```

```
                        fs=8000;
                    end
                end

                N=60;
                M=fix(0.02/(1/fs)); %tamaño de ventana para la
obtencion del pitch
                FI=6; %factor de interpolacion
                cortes=0;
                nota_ant=0;

                voz=evalin('base','voz');
                if voz
                    fc1=70;
                    fc2=1000;
                else
                    fc1=500;
                    fc2=5000;
                end
                %filtrado de la banda de interes
                grabación=fpa(grabacion,fs,fc1);
                grabación=fpb(grabacion,fs,fc2);
                %segmentado

inicio_fin=detector_inicio_fin(grabacion,fs,N,umb_ruido);

derivada=interp(normaliza(diff(envelope_reducida(grabacion,N))),N/2);
                for i=1:length(inicio_fin)
                    if length(inicio_fin{i,1})==2

tramo=grabación(inicio_fin{i,1}(1):inicio_fin{i,1}(2));
                        %obtener velocity como la derivada de primer
orden de la
                        %envolvente


vel=get_vel(max(derivada(inicio_fin{i,1}(1):inicio_fin{i,1}(2))),minim
o,maximo);
                        L=length(tramo);
                        k=1;
                        f=zeros(1,fix(L/M));
                        %obtencion de la variacion de F0 a lo largo de
la
                        %nota
                        for j=1:M:L-M*(L/M-fix(L/M))
                            ventana=tramo(j:j+M-1);
                            f(k)=pitch_xcorr(ventana,fs,FI);
                            k=k+1;
                        end
                        if ~isempty(f)
                            if length(f)>3
                                f=filtro_mediana(f);
                                f0=mode(f);
                            else
                                f0=mode(f);

                            end
                            if ~pb_on
```

```
                                if nota_ant==0 %si es la primera nota
en captarse

[numero_nota,nota_letra,nota_octava,~,~]=frec2midi(f0,fs*FI);
                                else %si no es la primera nota

intervalo=round(12*log2(f0/f_ant)); %se obtiene el intervalo
                                    numero_nota=nota_ant+intervalo;

nota_letra=instrumento2nota(numero_nota);
                                end
                                nota_ant=numero_nota;
                                f_ant=f0;

                                %escribir mensaje note on
                                switch acorde
                                    case 'no_chord'
                                        escribe_fichero(file,'On
Note', ...

inicio_fin{i,1}(1)/fs,nota_letra,nota_octava+octavar,vel);
                                        escribe_fichero(file,'Off
Note', ...

inicio_fin{i,1}(2)/fs,nota_letra,nota_octava+octavar,100);

                                    otherwise

letras=get_letra_acorde(nota_letra,acorde);
                                        escribe_fichero(file,'On
Note', ...

inicio_fin{i,1}(1)/fs,cell2mat(letras(1)),nota_octava+octavar,vel);
                                        escribe_fichero(file,'On
Note', ...

inicio_fin{i,1}(1)/fs,cell2mat(letras(2)),nota_octava+octavar,vel);
                                        escribe_fichero(file,'On
Note',...

inicio_fin{i,1}(1)/fs,cell2mat(letras(3)),nota_octava+octavar,vel);
                                        escribe_fichero(file,'Off
Note',...

inicio_fin{i,1}(2)/fs,cell2mat(letras(1)),nota_octava+octavar,100);
                                        escribe_fichero(file,'Off
Note',...

inicio_fin{i,1}(2)/fs,cell2mat(letras(2)),nota_octava+octavar,100);
                                        escribe_fichero(file,'Off
Note',...

inicio_fin{i,1}(2)/fs,cell2mat(letras(3)),nota_octava+octavar,100);

                                end
                                % creamos la grafica para mostrar en
la GUI
                                cortes(inicio_fin{i,1}(1))=vel;
                                cortes(inicio_fin{i,1}(2))=-80;

                            else
```

```
[~,nota_letra,nota_octava,~,f_nota_midi]=frec2midi(median(f),fs*FI);
                                escribe_fichero(file,'On Note',...

inicio_fin{i,1}(1)/fs,nota_letra,nota_octava+octavar,vel);


                                bend=pitch_bend(0,f(1),64,vel_bend);
                                escribe_fichero(file,'Pitch Wheel',...

inicio_fin{i,1}(1)/fs,nota_letra,nota_octava+octavar,128*(bend-64));


                                for k=2:length(f)
                                    %obtener desviacion
                                    bend=pitch_bend(f(k-
1),f(k),bend,vel_bend);
                                    %escribir el mensaje en el fichero
                                    escribe_fichero(file,'Pitch
Wheel',...

(inicio_fin{i,1}(1)+(k)*M)/fs,nota_letra,nota_octava+octavar,128*(bend
-64));
                                end
                                escribe_fichero(file,'Off Note',...

inicio_fin{i,1}(2)/fs,nota_letra,nota_octava+octavar,vel);
                                bend=64;
                                escribe_fichero(file,'Pitch Wheel',...

inicio_fin{i,1}(2)/fs,nota_letra,nota_octava+octavar,128*(bend-64));

                                cortes(inicio_fin{i,1}(1))=vel;
                                cortes(inicio_fin{i,1}(2))=-80;
                            end
                        end
                    end
                end
                %cerrar fichero
                fprintf(file,'  | End of track |  ');
                fclose(file);

                %se muestra el resultado en la GUI
                grabación=80*normaliza(grabación);
                stem(cortes,'r');grid
                set(gca,'ylim',[-90,140]);
                hold on
                plot(grabacion);
                % se habilita el boton de recalcular
                set(handles.recalcular,'Enable','on');
                hold off;
            end
        end
    end
end
```

```
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)

stop(handles.r);



% --- Executes on selection change in Sintetizadores.
function Sintetizadores_Callback(hObject, eventdata, handles)

disp_midi_elegido=get(handles.Sintetizadores,'Value');
assignin('base','disp_midi_elegido',disp_midi_elegido);



% --- Executes during object creation, after setting all properties.
function Sintetizadores_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in Instrumentos.
function Instrumentos_Callback(hObject, eventdata, handles)
instrumento=get(handles.Instrumentos,'Value');
assignin('base','instrumento',instrumento-1);

% --- Executes during object creation, after setting all properties.
function Instrumentos_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function Octavar_Callback(hObject, eventdata, handles)
octava=get(handles.Octavar,'String');
octava=str2num(octava);
assignin('base','octava',octava);

% --- Executes during object creation, after setting all properties.
function Octavar_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function umb_energia_Callback(hObject, eventdata, handles)
umb_energia=get(handles.umb_energia,'Value');
assignin('base','umb_energia',umb_energia);


% --- Executes during object creation, after setting all properties.
function umb_energia_CreateFcn(hObject, eventdata, handles)
```

```
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



% --- Executes on slider movement.
function vel_cambio_pitch_bend_Callback(hObject, ~, handles)
vel_bend=get(handles.vel_cambio_pitch_bend,'Value');
set(handles.valor_vel_cambio,'String',vel_bend);
assignin('base','vel_bend',vel_bend);



% --- Executes during object creation, after setting all properties.
function vel_cambio_pitch_bend_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



% --- Executes on slider movement.
function Umbral_energia_Callback(hObject, eventdata, handles)
valor=get(handles.Umbral_energia,'Value');
umbral_automatico=evalin('base','umb_energia_auto');
umb_energia=umbral_automatico+((0.005-umbral_automatico/10)/6)*(valor-
2);
if umb_energia<0
    umb_energia=umbral_automatico/10;
end
% umb_energia=umbral_automatico+valor*1e-3;

set(handles.valor_umbral_deteccion,'String',umb_energia);
assignin('base','umb_energia',umb_energia);




% --- Executes during object creation, after setting all properties.
function Umbral_energia_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function apagar_notas(obj,event)
import javax.sound.midi.*;
ai=obj;
data=ai.UserData;
if ~data.real_time && data.escribir

else
    switch data.acordes
        case 'no_chord'
            if data.real_time
                data.mens.setMessage(ShortMessage.NOTE_OFF,
data.channel,...
                    data.nota_ant ,100);
```

```
                data.midi.recep.send(data.mens, -1);
            end

            if data.escribir %si se ha habilitado la escritura en
fichero
                escribe_fichero(data.file,'Off Note',(data.contador-
1)*data.sig/data.fs,...

data.nota_letra,data.nota_octava+data.octavar,120);
            end


        case 'major'
            if data.real_time
                data.mens.setMessage(ShortMessage.NOTE_OFF,...
                    data.channel, data.nota_ant ,100);
                data.midi.recep.send(data.mens, -1);

                data.mens.setMessage(ShortMessage.NOTE_OFF,...
                    data.channel, data.nota_ant+4 ,100);
                data.midi.recep.send(data.mens, -1);

                data.mens.setMessage(ShortMessage.NOTE_OFF,...
                    data.channel, data.nota_ant+7 ,100);
                data.midi.recep.send(data.mens, -1);
            end



        case 'minor'
            if data.real_time
                data.mens.setMessage(ShortMessage.NOTE_OFF,...
                    data.channel, data.nota_ant ,100);
                data.midi.recep.send(data.mens, -1);

                data.mens.setMessage(ShortMessage.NOTE_OFF,...
                    data.channel, data.nota_ant+3 ,100);
                data.midi.recep.send(data.mens, -1);

                data.mens.setMessage(ShortMessage.NOTE_OFF,...
                    data.channel, data.nota_ant+7 ,100);
                data.midi.recep.send(data.mens, -1);
            end
        case 'power_chord'
            if data.real_time
                data.mens.setMessage(ShortMessage.NOTE_OFF,...
                    data.channel, data.nota_ant ,100);
                data.midi.recep.send(data.mens, -1);

                data.mens.setMessage(ShortMessage.NOTE_OFF,...
                    data.channel, data.nota_ant+5 ,100);
                data.midi.recep.send(data.mens, -1);

                data.mens.setMessage(ShortMessage.NOTE_OFF,...
                    data.channel, data.nota_ant+12 ,100);
                data.midi.recep.send(data.mens, -1);
            end
    end
end
```

```
%si cierra el fchero si procede
if data.escribir
    fprintf(data.file,'       | End of track |');
    fclose(data.file);
end


function analisis_reproduccion(obj,event)
import javax.sound.midi.*;

%umbrales para detección
umb_zcr=0.15;
umb_max_vel=0.005;


ai=obj;
data=ai.UserData;

datos_nuevos=getdata(ai); %cogemos datos
if data.escribir && ~data.real_time %si se desea escribir en fichero
pero no reproducir en tiempo real, se activa el modo offline
    data.temporal(1:(data.contador-
1)*data.sig)=[data.temporal,datos_nuevos'];
else
    datos_nuevos=fpa(datos_nuevos,data.fs,data.rango(1)); %filtrado
    max(diff(datos_nuevos));

[estado,energ_act]=det_estado(datos_nuevos,data.energ_max,umb_zcr,data
.umb_energ,data.umb_derivada,data.estado_anterior,data.signal);
    if energ_act>data.energ_max %comparar energia actual con maxima,
para almacenar la maxima
        data.energ_max=energ_act;
    end
    disp(estado);
    switch estado
        case 'inicio'
            data.energ_max=0;
            data.mens.setMessage(ShortMessage.PITCH_BEND,
data.channel, 64);data.bend=64;
            data.midi.recep.send(data.mens, -1);

        case 'sostenido'
            ventana=[data.ventana_anterior;datos_nuevos];
            data.f_act_vent=pitch_xcorr(ventana,data.fs,data.FI);
            vel=get_vel(energ_act,data.umb_energ,umb_max_vel);
            if strcmp(data.estado_anterior,'inicio')
                if data.rango(1)<data.f_act_vent &&
data.f_act_vent<data.rango(2)


[data.nota,data.nota_letra,data.nota_octava,desviacion,~]...
                        =frec2midi(data.f_act_vent,data.fs*data.FI);
                    vel=get_vel(energ_act,data.umb_energ,umb_max_vel);

                    switch data.acordes
                        case 'no_chord'
                            if data.pb_on

data.bend=pitch_bend(0,desviacion,data.bend,data.vel_bend);
```

```
data.mens.setMessage(ShortMessage.PITCH_BEND, data.channel,
data.bend);
                                data.midi.recep.send(data.mens, -1);
                            end
                            if data.escribir==1 %si se ha habilitado
la escritura en fichero
                                escribe_fichero(data.file,'On
Note',...
                                    (data.contador-
1)*data.sig/data.fs,data.nota_letra,data.nota_octava+data.octavar,vel)
;
                            end



                            if data.real_time
                                data.nota=data.nota+12*data.octavar;

data.mens.setMessage(ShortMessage.NOTE_ON,...
                                    data.channel, data.nota, vel);
                                data.midi.recep.send(data.mens, -1);
                            end

                        case 'major'
                            data.acorde{1,1}=data.nota_letra;
                            data.acorde{1,2}=data.nota_octava;

[data.acorde{2,1},data.acorde{2,2},data.acorde{3,1}...

,data.acorde{3,2}]=acorde(data.nota_letra,data.acordes);
                            data.nota=data.nota+12*data.octavar;
                            if data.real_time

data.mens.setMessage(ShortMessage.NOTE_ON,...
                                    data.channel, data.nota, vel);
                                data.midi.recep.send(data.mens, -1);


data.mens.setMessage(ShortMessage.NOTE_ON,...
                                    data.channel, data.nota+4, vel);
                                data.midi.recep.send(data.mens, -1);


data.mens.setMessage(ShortMessage.NOTE_ON,...
                                    data.channel, data.nota+7, vel);
                                data.midi.recep.send(data.mens, -1);
                            end

                            if data.escribir
                                escribe_fichero(data.file,'On
Note',...
                                    (data.contador-
1)*data.sig/data.fs,data.acorde{1,1},data.acorde{1,2},vel);
                                escribe_fichero(data.file,'On
Note',...
                                    (data.contador-
1)*data.sig/data.fs,data.acorde{2,1},data.acorde{2,2}+data.acorde{1,2}
,vel);
                                escribe_fichero(data.file,'On
Note',...
```

```
                                    (data.contador-
1)*data.sig/data.fs,data.acorde{3,1},data.acorde{3,2}+data.acorde{1,2}
,vel);
                            end
                        case 'minor'
                            data.nota=data.nota+12*data.octavar;

                            if data.real_time

data.mens.setMessage(ShortMessage.NOTE_ON,...
                                    data.channel, data.nota, vel);
                                data.midi.recep.send(data.mens, -1);


data.mens.setMessage(ShortMessage.NOTE_ON,...
                                    data.channel, data.nota+3, vel);
                                data.midi.recep.send(data.mens, -1);


data.mens.setMessage(ShortMessage.NOTE_ON,...
                                    data.channel, data.nota+7, vel);
                                data.midi.recep.send(data.mens, -1);
                            end



                            if data.escribir
                                data.acorde{1,1}=data.nota_letra;
                                data.acorde{1,2}=data.nota_octava;

data.acorde=get_letra_acorde(data.nota_letra,data.acordes);
                                escribe_fichero(data.file,'On
Note',...
                                    (data.contador-
1)*data.sig/data.fs,data.acorde{1,1},data.acorde{1,2},vel);
                                escribe_fichero(data.file,'On
Note',...
                                    (data.contador-
1)*data.sig/data.fs,data.acorde{2,1},data.acorde{2,2}+data.acorde{1,2}
,vel);
                                escribe_fichero(data.file,'On
Note',...
                                    (data.contador-
1)*data.sig/data.fs,data.acorde{3,1},data.acorde{3,2}+data.acorde{1,2}
,vel);
                            end



                        case 'power_chord'
                            data.nota=data.nota+12*data.octavar;

                            data.acorde{1,1}=data.nota_letra;
                            data.acorde{1,2}=data.nota_octava;

[data.acorde{2,1},data.acorde{2,2},data.acorde{3,1},...

data.acorde{3,2}]=acorde(data.nota_letra,data.acordes);

                            if data.real_time

data.mens.setMessage(ShortMessage.NOTE_ON,...
```

```
                                    data.channel, data.nota, vel);
                                data.midi.recep.send(data.mens, -1);


data.mens.setMessage(ShortMessage.NOTE_ON,...
                                    data.channel, data.nota+5, vel);
                                data.midi.recep.send(data.mens, -1);


data.mens.setMessage(ShortMessage.NOTE_ON,...
                                    data.channel, data.nota+12, vel);
                                data.midi.recep.send(data.mens, -1);
                            end

                            if data.escribir
                                escribe_fichero(data.file,'On
Note',...
                                    (data.contador-
1)*data.sig/data.fs,...

data.acorde{1,1},data.acorde{1,2},vel);
                                escribe_fichero(data.file,'On
Note',...
                                    (data.contador-
1)*data.sig/data.fs,...

data.acorde{2,1},data.acorde{2,2}+data.acorde{1,2},vel);
                                escribe_fichero(data.file,'On
Note',...
                                    (data.contador-
1)*data.sig/data.fs,...

data.acorde{3,1},data.acorde{3,2}+data.acorde{1,2},vel);
                            end
                    end

                else
                    estado='ruido';
                end
            else


                if data.f_ant_vent~=data.f_act_vent
                    if data.pb_on

data.bend=pitch_bend(data.f_ant_vent,data.f_act_vent,data.bend,data.ve
l_bend);
                    end

                    if data.real_time
                        if data.pb_on

data.mens.setMessage(ShortMessage.PITCH_BEND, data.channel,
data.bend);
                            data.midi.recep.send(data.mens, -1);
                        end


data.mens.setMessage(ShortMessage.CHANNEL_PRESSURE, data.channel,
vel);
```

```
                        data.midi.recep.send(data.mens, -1);
                    end

                    if data.escribir && data.pb_on
                        escribe_fichero(data.file,'Pitch Wheel',...
                            (data.contador-
1)*data.sig/data.fs,data.nota_letra,...
                            data.nota_octava,128*(data.bend-64));
                    end
                end
            end

        case 'off'
            switch data.acordes
                case 'no_chord'
                    if data.real_time
                        data.mens.setMessage(ShortMessage.NOTE_OFF,...
                            data.channel, data.nota_ant ,100);
                        data.midi.recep.send(data.mens, -1);
                    end


                    if data.escribir==1 %si se ha habilitado la
escritura en fichero
                        escribe_fichero(data.file,'Off
Note',(data.contador-1)*data.sig/data.fs,...

data.nota_letra,data.nota_octava+data.octavar,120);
                        if data.pb_on
                            escribe_fichero(data.file,'Pitch
Wheel',(data.contador-1)*data.sig/data.fs,...

data.nota_letra,data.nota_octava+data.octavar,0);
                        end
                    end



                case 'major'
                    if data.real_time
                        data.mens.setMessage(ShortMessage.NOTE_OFF,...
                            data.channel, data.nota_ant ,100);
                        data.midi.recep.send(data.mens, -1);

                        data.mens.setMessage(ShortMessage.NOTE_OFF,...
                            data.channel, data.nota_ant+4 ,100);
                        data.midi.recep.send(data.mens, -1);

                        data.mens.setMessage(ShortMessage.NOTE_OFF,...
                            data.channel, data.nota_ant+7 ,100);
                        data.midi.recep.send(data.mens, -1);
                    end



                    data.energ_max=0;
                    data.mens.setMessage(ShortMessage.PITCH_BEND,...
                        data.channel, 64);data.bend=64;
                    data.midi.recep.send(data.mens, -1);
```

```
                    if data.escribir
                        escribe_fichero(data.file,'Off Note',...
                            (data.contador-1)*data.sig/data.fs,...
                            data.acorde{1,1},data.acorde{1,2},120);
                        escribe_fichero(data.file,'Off Note',...
                            (data.contador-1)*data.sig/data.fs,...
data.acorde{2,1},data.acorde{2,2}+data.acorde{1,2},120);
                        escribe_fichero(data.file,'Off Note',...
                            (data.contador-1)*data.sig/data.fs,...
data.acorde{3,1},data.acorde{3,2}+data.acorde{1,2},120);

                        if data.pb_on
                            escribe_fichero(data.file,'Pitch
Wheel',...
                                (data.contador-
1)*data.sig/data.fs,data.nota_letra,data.nota_octava+data.octavar,0);
                        end

                    end
                case 'minor'

[data.nota,~,~,~]=frec2midi(data.f_act_vent,data.fs);
                    data.nota=data.nota+12*data.octavar;
                    if data.real_time
                        data.mens.setMessage(ShortMessage.NOTE_OFF,...
                            data.channel, data.nota_ant, 100);
                        data.midi.recep.send(data.mens, -1);

                        data.mens.setMessage(ShortMessage.NOTE_OFF,...
                            data.channel, data.nota_ant+3, 100);
                        data.midi.recep.send(data.mens, -1);

                        data.mens.setMessage(ShortMessage.NOTE_OFF,...
                            data.channel, data.nota_ant+7, 100);
                        data.midi.recep.send(data.mens, -1);
                    end



                    data.energ_max=0;
                    data.mens.setMessage(ShortMessage.PITCH_BEND,
data.channel, 64);
                    data.midi.recep.send(data.mens, -1);
                    data.bend=64;

                    if data.escribir
                        escribe_fichero(data.file,'Off Note',...
                            (data.contador-
1)*data.sig/data.fs,data.acorde{1,1},data.acorde{1,2},120);
                        escribe_fichero(data.file,'Off Note',...
                            (data.contador-
1)*data.sig/data.fs,data.acorde{2,1},data.acorde{2,2}+data.acorde{1,2}
,120);
                        escribe_fichero(data.file,'Off Note',...
                            (data.contador-
1)*data.sig/data.fs,data.acorde{3,1},data.acorde{3,2}+data.acorde{1,2}
,120);
                        if data.pb_on
```

```
                            escribe_fichero(data.file,'Pitch
Wheel',...
                                (data.contador-
1)*data.sig/data.fs,data.nota_letra,data.nota_octava+data.octavar,0);
                        end
                    end
                case 'power_chord'

[data.nota,~,~,~]=frec2midi(data.f_act_vent,data.fs);
                    data.nota=data.nota+12*data.octavar;
                    if data.real_time
                        data.mens.setMessage(ShortMessage.NOTE_OFF,...
                            data.channel, data.nota_ant, 100);
                        data.midi.recep.send(data.mens, -1);

                        data.mens.setMessage(ShortMessage.NOTE_OFF,...
                            data.channel, data.nota_ant+5, 100);
                        data.midi.recep.send(data.mens, -1);

                        data.mens.setMessage(ShortMessage.NOTE_OFF,...
                            data.channel, data.nota_ant+12, 100);
                        data.midi.recep.send(data.mens, -1);
                    end


                    data.energ_max=0;
                    if data.pb_on
                        data.mens.setMessage(ShortMessage.PITCH_BEND,
data.channel, 64);
                        data.midi.recep.send(data.mens, -1);
                    end

                    if data.escribir
                        escribe_fichero(data.file,'Off Note',...
                            (data.contador-
1)*data.sig/data.fs,data.acorde{1,1},data.acorde{1,2},120);
                        escribe_fichero(data.file,'Off Note',...
                            (data.contador-
1)*data.sig/data.fs,data.acorde{2,1},data.acorde{2,2}+data.acorde{1,2}
,120);
                        escribe_fichero(data.file,'Off Note',...
                            (data.contador-
1)*data.sig/data.fs,data.acorde{3,1},data.acorde{3,2}+data.acorde{1,2}
,120);
                        if data.pb_on
                            escribe_fichero(data.file,'Pitch
Wheel',...
                                (data.contador-
1)*data.sig/data.fs,data.nota_letra,data.nota_octava+data.octavar,0);
                        end

                    end
            end
        case 'ruido'
            data.bend=64;
    end

    data.ventana_anterior=datos_nuevos;
    data.nota_ant=data.nota;
    data.f_ant_vent=data.f_act_vent;
```

```
    data.estado_anterior=estado;
end
data.contador=data.contador+1;
set(obj,'UserData',data);



% --- Executes on button press in major.
function major_Callback(hObject, eventdata, handles)
assignin('base','acorde','major');

% --- Executes on button press in minor.
function minor_Callback(hObject, eventdata, handles)
assignin('base','acorde','minor');

% --- Executes on button press in power_chord.
function power_chord_Callback(hObject, eventdata, handles)
assignin('base','acorde','power_chord');

% --- Executes when selected object is changed in acordes.
function acordes_SelectionChangeFcn(hObject, eventdata, handles)



% --- Executes on button press in no_chord.
function no_chord_Callback(hObject, eventdata, handles)
assignin('base','acorde','no_chord');



% --- Executes on button press in Escribir.
function Escribir_Callback(hObject, eventdata, handles)

if (get(hObject,'Value') == get(hObject,'Max'))

    assignin('base','escritura_habilitada',1);
    set(handles.busca_fichero,'Enable','on');
    set(handles.Escribir,'Enable','on','Value',1);
    set(handles.real_time,'Value',0);

else
    assignin('base','escritura_habilitada',0);
    set(handles.busca_fichero,'Enable','off');
    set(handles.real_time,'Enable','on');
    % Checkbox is not checked-take appropriate action
end



function nombre_fichero_escritura_Callback(hObject, eventdata,
handles)

% --- Executes during object creation, after setting all properties.
function nombre_fichero_escritura_CreateFcn(hObject, eventdata,
handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on button press in busca_fichero.
function busca_fichero_Callback(hObject, eventdata, handles)
[nombre_fichero,path_fichero] = uiputfile({'*.txt','Archivo de texto
(*.txt)'},'Nuevo archivo');

assignin('base','fichero_path_nombre',[path_fichero nombre_fichero]);


% --- Executes during object creation, after setting all properties.
function acordes_CreateFcn(hObject, eventdata, handles)

function valor_umbral_deteccion_ButtonDownFcn(hObject, eventdata,
handles)

% --- Executes during object creation, after setting all properties.
function valor_umbral_deteccion_CreateFcn(hObject, eventdata, handles)


% --- Executes on button press in real_time.
function real_time_Callback(hObject, eventdata, handles)
if (get(hObject,'Value') == get(hObject,'Max'))

    assignin('base','real_time',1);
    set(handles.Escribir,'Value',0);

else
    assignin('base','real_time',0);
    set(handles.Escribir,'Enable','on');

    % Checkbox is not checked-take appropriate action
end

% Hint: get(hObject,'Value') returns toggle state of real_time


% --- Executes during object creation, after setting all properties.
function imagen_CreateFcn(hObject, eventdata, handles)

% --- Executes on button press in habilitar_bend.
function habilitar_bend_Callback(hObject, eventdata, handles)
if (get(hObject,'Value') == get(hObject,'Max'))

    set(handles.text3,'Enable','on');
    set(handles.habilitar_bend,'Value',1);
    set(handles.vel_cambio_pitch_bend,'Enable','on');
    assignin('base','pb_on',1);
else
    set(handles.text3,'Enable','off');
    set(handles.habilitar_bend,'Value',0);
    set(handles.vel_cambio_pitch_bend,'Enable','off');
    assignin('base','pb_on',0);
    % Checkbox is not checked-take appropriate action
end


% --- Executes when selected object is changed in uipanel7.
function uipanel7_SelectionChangeFcn(hObject, eventdata, handles)
if strcmp(get(hObject,'String'),'Voz')
```

```
    assignin('base','voz',1);
    %     assignin('base','silbido',0);
else
    %     assignin('base','silbido',1);
    assignin('base','voz',0);
end

% --- Executes on key press with focus on power_chord and none of its
controls.
function power_chord_KeyPressFcn(hObject, eventdata, handles)

% --- Executes on button press in voz.
function voz_Callback(hObject, eventdata, handles)

% --- Executes on key press with focus on voz and none of its
controls.
function voz_KeyPressFcn(hObject, eventdata, handles)
assignin('base','voz',1);


% --- Executes during object creation, after setting all properties.
function Imagen_CreateFcn(hObject, eventdata, handles)
imshow('Memoria\Imagenes\Icono Funcion melodica.jpg');
% Hint: place code in OpeningFcn to populate imagen


% --- Executes on button press in recalcular.
function recalcular_Callback(hObject, eventdata, handles)
umb_ruido=evalin('base','umb_energia');
vel_bend=evalin('base','vel_bend');
octavar=evalin('base','octava');
grabacion=evalin('base','grabacion');
pb_on=evalin('base','pb_on');
acorde=evalin('base','acorde');
nombre_fichero=evalin('base','fichero_path_nombre');
BPM=evalin('base','bpm');
file = fopen(nombre_fichero,'w');
fprintf(file,['MThd | # of Tracks=1 | '...
    'Division=1000 \n\n Track #1 \n       | Tempo | BPM= %s \n'],BPM);
fs=8000;
N=60;
M=fix(0.02/(1/fs)); %tamaño de ventana para la obtencion del pitch
FI=6; %factor de interpolacion
cortes=0;
nota_ant=0;

máximo=1;
minimo=evalin('base','minimo');
voz=evalin('base','voz');

if voz
    fc1=70;
    fc2=1000;
else
    fc1=500;
    fc2=5000;
end
%filtrado de la banda
grabación=fpa(grabacion,fs,fc1);
grabación=fpb(grabacion,fs,fc2);
```

```
%segmentado de eventos
inicio_fin=detector_inicio_fin(grabacion,fs,N,umb_ruido);
derivada=interp(normaliza(diff(envelope_reducida(grabacion,N))),N/2);
for i=1:length(inicio_fin)
    if length(inicio_fin{i,1})==2
        tramo=grabación(inicio_fin{i,1}(1):inicio_fin{i,1}(2));
        %obtener velocity como la derivada de primer orden de la
        %envolvente


vel=get_vel(max(derivada(inicio_fin{i,1}(1):inicio_fin{i,1}(2))),minim
o,maximo);
        L=length(tramo);
        k=1;
        f=zeros(1,fix(L/M));
        %calculo de la variacion de F0 a lo largo de la nota
        for j=1:M:L-M*(L/M-fix(L/M))
            ventana=tramo(j:j+M-1);
            f(k)=pitch_xcorr(ventana,fs,FI);
            k=k+1;
        end
        if ~isempty(f)
            if length(f)>3
                f=filtro_mediana(f);
                f0=mode(f);
            else
                f0=mode(f);

            end

            if ~pb_on %si no se va a mandar mensajes de pitch bend

                if nota_ant==0 %si es la primera nota en captarse

[numero_nota,nota_letra,nota_octava,~,~]=frec2midi(f0,fs*FI);
                else %si no es la primera nota
                    intervalo=round(12*log2(f0/f_ant)); %se obtiene el
intervalo
                    numero_nota=nota_ant+intervalo;
                    nota_letra=instrumento2nota(numero_nota);
                end
                nota_ant=numero_nota;
                f_ant=f0;

                %escribir mensaje note on
                switch acorde
                    case 'no_chord'
                        escribe_fichero(file,'On Note',...

inicio_fin{i,1}(1)/fs,nota_letra,nota_octava+octavar,vel);
                        escribe_fichero(file,'Off Note',...

inicio_fin{i,1}(2)/fs,nota_letra,nota_octava+octavar,100);

                    otherwise
                        letras=get_letra_acorde(nota_letra,acorde);
                        escribe_fichero(file,'On Note',...

inicio_fin{i,1}(1)/fs,cell2mat(letras(1)),nota_octava+octavar,vel);
                        escribe_fichero(file,'On Note',...
```

```
inicio_fin{i,1}(1)/fs,cell2mat(letras(2)),nota_octava+octavar,vel);
                        escribe_fichero(file,'On Note',...

inicio_fin{i,1}(1)/fs,cell2mat(letras(3)),nota_octava+octavar,vel);
                        escribe_fichero(file,'Off Note',...

inicio_fin{i,1}(2)/fs,cell2mat(letras(1)),nota_octava+octavar,100);
                        escribe_fichero(file,'Off Note',...

inicio_fin{i,1}(2)/fs,cell2mat(letras(2)),nota_octava+octavar,100);
                        escribe_fichero(file,'Off Note',...

inicio_fin{i,1}(2)/fs,cell2mat(letras(3)),nota_octava+octavar,100);

                end
                % creamos la grafica para mostrar en la GUI
                cortes(inicio_fin{i,1}(1))=vel;
                cortes(inicio_fin{i,1}(2))=-80;

            else

[~,nota_letra,nota_octava,~,f_nota_midi]=frec2midi(median(f),fs*FI);
                escribe_fichero(file,'On
Note',inicio_fin{i,1}(1)/fs,nota_letra,nota_octava+octavar,vel);


                bend=pitch_bend(0,f(1),64,vel_bend);
                escribe_fichero(file,'Pitch
Wheel',inicio_fin{i,1}(1)/fs...
                    ,nota_letra,nota_octava+octavar,128*(bend-64));


                for k=2:length(f)
                    %obtener desviacion
                    bend=pitch_bend(f(k-1),f(k),bend,vel_bend);
                    %escribir el mensaje en el fichero
                    escribe_fichero(file,'Pitch Wheel',...
                        (inicio_fin{i,1}(1)+(k)*M)/fs,nota_letra,...
                        nota_octava+octavar,128*(bend-64));
                end
                escribe_fichero(file,'Off
Note',inicio_fin{i,1}(2)/fs,...
                    nota_letra,nota_octava+octavar,vel);
                bend=64;
                escribe_fichero(file,'Pitch
Wheel',inicio_fin{i,1}(2)/fs,...
                    nota_letra,nota_octava+octavar,128*(bend-64));

                cortes(inicio_fin{i,1}(1))=vel;
                cortes(inicio_fin{i,1}(2))=-80;
            end
        end
    end
end
%cerrar fichero
fprintf(file,'       | End of track |\n');
fclose(file);

grabación=80*normaliza(grabación);
```

```
%se muestran los resultados por GUI
set(gca,'ylim',[-90,140]);
stem(cortes,'r');hold on;
plot(grabacion);
grid
hold off;


% --- Executes on slider movement.
function velocity_Callback(hObject, eventdata, handles)
valor=get(handles.velocity,'Value');
minimo=0;
máximo=1;
y=(minimo)+(maximo-minimo)*(valor);
assignin('base','minimo',y);


% --- Executes during object creation, after setting all properties.
function velocity_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in importar.
function importar_Callback(hObject, eventdata, handles)
if (get(hObject,'Value') == get(hObject,'Max'))

    assignin('base','importar',1);
    set(handles.importar_browse,'Enable','on');
    assignin('base','microfono',0);
    set(handles.microfono,'Value',0);
    assignin('base','real_time',0);
    set(handles.Escribir,'Value',1,'Enable','on');
    assignin('base','escritura_habilitada',1);
    set(handles.real_time,'Enable','off','Value',0);
    assignin('base','control_inicio',0);
else
    assignin('base','importar',0);
    set(handles.importar_browse,'Enable','off');
    set(handles.real_time,'Enable','on');
    assignin('base','control_inicio',1);


    % Checkbox is not checked-take appropriate action
end

% Hint: get(hObject,'Value') returns toggle state of importar


% --- Executes on button press in importar_browse.
function importar_browse_Callback(hObject, eventdata, handles)
[nombre_fichero,path_fichero] = uigetfile({'*.wav','Archivo de
audio(*.wav)'},'Nuevo archivo');
if path_fichero~=0
    [grabacion]=wavread([path_fichero nombre_fichero]);
    assignin('base','grabacion',grabacion);
```

```
    set(handles.Iniciar,'Enable','on');
end

% --- Executes on button press in microfono.
function microfono_Callback(hObject, eventdata, handles)
if (get(hObject,'Value') == get(hObject,'Max'))

    assignin('base','microfono',1);
    assignin('base','importar',0);
    set(handles.importar,'Value',0);
    set(handles.importar_browse,'Enable','off');
    set(handles.real_time,'Enable','on');
    set(handles.Iniciar,'Enable','on');
else
    assignin('base','microfono',0);
    set(handles.Iniciar,'Enable','off');
    % Checkbox is not checked-take appropriate action
end

% Hint: get(hObject,'Value') returns toggle state of microfono



function bpm_Callback(hObject, eventdata, handles)
bpm=get(handles.bpm,'String');
assignin('base','bpm',bpm);
% Hints: get(hObject,'String') returns contents of bpm as text
%        str2double(get(hObject,'String')) returns contents of bpm as
a double


% --- Executes during object creation, after setting all properties.
function bpm_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function escribe_fichero(file,mensaje,segundo_evento,
nota,octava,velocity)
% ------------------------------------------------------------------
% escribe_fichero  Escritura de evento en fichero de midi texto
%   escribe(FILE,MENSAJE,SEGUNDO_EVENTO,NOTA,OCTAVA,VELOCITY) escribe
en
%   el fichero con handle FILE, el evento definido en MENSAJE, que se
%   produce en el sengundo SEGUNDO_EVENTO, con nota midi NOTA, octava
%   OCTAVA y velocidad (volumen) VELOCITY.
%
%   MENSAJE: "On Note"
%            "Off Note"
%            "Pitch Wheel"
%            "Aftertouch"
%   NOTA: nota midi en notacion americana. Ejemplo A#3 La sostenido de
la
%   tercera octava
%   OCTAVA: Numero entero entre -2 y 8.
%   VELOCITY: Valor de el volumen del evento entre 0 y 127;
% ------------------------------------------------------------------

natural=fix(segundo_evento);
bloque=fix(natural/4);
segundos=natural-bloque*4;%4 compases se consideran un bloque
decimales=segundo_evento-fix(segundo_evento);
decimales_todos=(decimales);
decimales=(num2str(decimales_todos));
L=length(decimales);
if length(decimales)>5
    decimales=decimales(1:5);
end
L=length(decimales);
switch L

    case 3
        mil=[decimales(3),'0','0'];
    case 4
        mil=[decimales(3),decimales(4),'0'];
    case 5
        if decimales_todos>=5 && decimales_todos(6)>5
            decimales(5)=decimales(5)+1;
        end
        mil=[decimales(3),decimales(4),decimales(5)];
    otherwise
        mil=['0','0','0'];

end
bloque=bloque+1;
segundos=segundos+1;

switch mensaje(1:2)
    case 'On'
        cadena = 'On Note   ';
    case 'Of'
        cadena = 'Off Note  ';
    case 'Af'
        cadena = 'Aftertouch';
    case 'Pi'
        cadena = 'Pitch Wheel';
```

```
end

fprintf(file,' %3d:%3d:%3s',bloque,segundos,mil);
fprintf(file,' |%8s |',cadena);
fprintf(file,' chan=1 | ');
if strcmp(mensaje(1:2),'Pi')
    fprintf(file,' bend= %3d \n',velocity);
else
    fprintf(file,' pitch= %2s %2d | ',nota,octava);
    if strcmp(mensaje(1:2),'Af')
        fprintf(file,'press= %3d \n',velocity);
    else
        fprintf(file,'vol=   %3d \n',velocity);
    end
end

function letras=get_letra_acorde(letra,acorde)
% ---------------------------------------------------------------------
%get_letra_acorde    obtencion de notas de acorde
%   LETRAS=get_letra_acorde(LETRA,ACORDE), almacena en LETRAS, las
%   diferentes notas en notacion americana, partiend desde la nota
LETRA, y
%   con intervalos correspondientes al acorde designado por ACORDE.
%
%   ACORDE: 'major' acorde mayor, intervalos 1-4-7
%           'minor' acorde menor, intervalos 1-3-7
%           'power_chord' acorde de poder, intervalos 1-5-12
% ---------------------------------------------------------------------
notas={'C';'C#';'D';'D#';'E';'F';'F#';'G';'G#';'A';'A#';'B';...
    'C';'C#';'D';'D#';'E';'F';'F#';'G';'G#''A';'A#';'B'};

%obtener octava
if ~strcmp(letra(2),' ') && ~strcmp(letra(2),'#')
    octava=letra(2);
else if ~strcmp(letra(3),' ') && ~strcmp(letra(3),'#')
        octava=letra(3);
    end
end
%obtener letras de la nota
l=letra;
if ~strcmp(l(1),' ')
    aux(1)=l(1);
else if ~strcmp(l(2),' ')
        aux(1)=l(2);
    end
end
L=length(l);
for i=1:L
    if strcmp('#',l(i))
        aux(2)='#';
    end
end
letra=aux;
l=letra;

indices=strcmp(notas,letra);
i=1;
while i<length(indices) && indices(i)==0
    i=i+1;
end
switch acorde
```

```
case 'major'
    if length(notas{i+4,1})==2
        if i+4>12
            letra2=[notas{i+4,1}(1,1),'#',octava+1];
        else
            letra2=[notas{i+4,1}(1,1),'#',octava];
        end
    else
        if i+4>12
            letra2=[notas{i+4,1}(1,1),octava+1];
        else
            letra2=[notas{i+4,1}(1,1),octava];
        end
    end

    if length(notas{i+7,1})==2
        if i+7>12
            letra3=[notas{i+7,1}(1,1),'#',octava+1];
        else
            letra3=[notas{i+7,1}(1,1),'#',octava];
        end
    else
        letra3=[notas{i+7,1}(1,1),octava];
    end
    letras={l,letra2,letra3};
case 'minor'
    if length(notas{i+3,1})==2
        if i+3>12
            letra2=[notas{i+3,1}(1,1),'#',octava+1];
        else
            letra2=[notas{i+3,1}(1,1),'#',octava];
        end
    else
        if i+3>12
            letra2=[notas{i+3,1}(1,1),octava+1];
        else
            letra2=[notas{i+3,1}(1,1),octava];
        end
    end

    if length(notas{i+7,1})==2
        if i+7>12
            letra3=[notas{i+7,1}(1,1),'#',octava+1];
        else
            letra3=[notas{i+7,1}(1,1),'#',octava];
        end
    else
        letra3=[notas{i+7,1}(1,1),octava];
    end
    letras={l,letra2,letra3};
case 'power_chord'
    if length(notas{i+5,1})==2
        if i+5>12
            letra2=[notas{i+5,1}(1,1),'#',octava+1];
        else
            letra2=[notas{i+5,1}(1,1),'#',octava];
        end
    else
        if i+5>12
            letra2=[notas{i+5,1}(1,1),octava+1];
        else
```

```
                letra2=[notas{i+5,1}(1,1),octava];
            end
        end

        letra3=l;
        if length(letra3)==3
            letra3(3)=octava+1;
        else
            letra3(2)=octava+1;
        end

        letras={[letra,octava],letra2,letra3};
end

function y=fpb(x,fs,fc)
%--------------------------------------------------------------------
%FPB    filtrado paso bajo
%   Y=fpb(X,FS,FC) almacena en Y la señal resultante de filtrar paso
bajo
%   con frecuencia de corte FC, la señal de entrada X de frecuencia de
%   muestreo FS.
%--------------------------------------------------------------------

wn=fc/fs;
N=20;%orden
%obtenemos coeficientes
B=fir1(N,wn);
%filtramos
x1=conv(B,x);
y=zeros(6,length(x1));
y=x1;

function y=fpa(x,fs,fc)
% --------------------------------------------------------------------
%FPA    filtrado paso alto
%   Y=fpb(X,FS,FC) almacena en Y la señal resultante de filtrar paso
alto
%   con frecuencia de corte FC, la señal de entrada X de frecuencia de
%   muestreo FS.
% --------------------------------------------------------------------
N=20;
wn=fc/fs;
% obtenemos coeficientes
B=fir1(N,wn,'high');
% filtramos
y=conv(B,x);
X=length(x);
Y=length(y);
D=fix((Y-X)/2);
y=y(D:D+X-1);
```

```
function y=filtro_mediana(x)
%-------------------------------------------------------------------
%FILTRO_MEDIANA    filtrado de mediana
%   Y=filtro_mediana(X), almacena en Y el valor de la señal X filtrada
con
%   un filtro unidimensional de mediana de tamaño de mascara 3.
%-------------------------------------------------------------------
X=length(x);
if X>3
    y=zeros(1,X);
    for i=1:X-2
        y(i)=median(x(i:i+2));
    end
    y(X-2:X)=x(X-2:X);
else
    y=x;
end

function y=envelope_reducida(x,N)
%ENVELOPE_REDUCIDA     Obtencion de la envolvente de una señal de
audio.
%   Y=envelope_reducida(X,N), obtiene la envolvente Y de la señal de
%   entrada X, tomando ventanas de N puntos con solapamiento del 50%.
Cada
%   valor de Y representa N/2 muestras de X.

M=length(x);
y=zeros(1,round(M/N));
j=1;
for i=N/2+1:N/2:M-N/2
    y(j)=max(abs(x(i-N/2:i+N/2)));
    j=j+1;
end

function signal_norm=normaliza(signal)
%-------------------------------------------------------------------
%NORMALIZA    Normalizacion.
%       SIGNAL_NORM=normaliza(SIGNAL), obtiene la señal SIGNAL_NORM
como
%       normalizacion entre los valores -1 y 1 de la señal de entrada
%       SIGNAL.
% -------------------------------------------------------------------
N=(max(abs(signal)));
signal_norm=signal./N;

function nota=instrumento2nota(instrumento)
%-------------------------------------------------------------------
%INSTRUMENTO2NOTA     Notación americana y octava
%   NOTA=instrumento2nota(INSTRUMENTO), almacena en NOTA el valor en
%   notacion americana de la nota correspondiente al instrumento (en
numero
%   MIDI) que se l de a la entrada.
%-------------------------------------------------------------------
n=[0 12 24 36 48 60 72 84 96 108 120];
if max(instrumento==0+n)
    [~,octava]=max(instrumento==0+n);
    octava=octava-2;
    nota=['C',num2str(octava)];
else if max(instrumento==1+n)
        [~,octava]=max(instrumento==1+n);
        octava=octava-2;
```

```
        nota=['C#',num2str(octava)];
    else if max(instrumento==2+n)
            [~,octava]=max(instrumento==2+n);
            octava=octava-2;
            nota=['D',num2str(octava)];
        else if max(instrumento==3+n)
                [~,octava]=max(instrumento==3+n);
                octava=octava-2;
                nota=['D#',num2str(octava)];
            else if max(instrumento==4+n)
                    [~,octava]=max(instrumento==4+n);
                    octava=octava-2;
                    nota=['E',num2str(octava)];
                else if max(instrumento==5+n)
                        [~,octava]=max(instrumento==5+n);
                        octava=octava-2;
                        nota=['F',num2str(octava)];
                    else if max(instrumento==6+n)
                            [~,octava]=max(instrumento==6+n);
                            octava=octava-2;
                            nota=['F#',num2str(octava)];
                        else if max(instrumento==7+n)
                                [~,octava]=max(instrumento==7+n);
                                octava=octava-2;
                                nota=['G',num2str(octava)];
                            else if max(instrumento==8+n)
                                    [~,octava]=max(instrumento==8+n);
                                    octava=octava-2;
                                    nota=['G#',num2str(octava)];
                                else if max(instrumento==9+n)

[~,octava]=max(instrumento==9+n);
                                        octava=octava-2;
                                        nota=['A',num2str(octava)];
                                    else if max(instrumento==10+n)

[~,octava]=max(instrumento==10+n);
                                            octava=octava-2;

nota=['A#',num2str(octava)];
                                        else if max(instrumento==11+n)

[~,octava]=max(instrumento==11+n);
                                                octava=octava-2;

nota=['B',num2str(octava)];
                                            end
                                        end
                                    end
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
function y=pitch_xcorr(x,fs,FI)
%-----------------------------------------------------------------------
-----
```

```
% PITCH_XCORR    obtencion de tono por medio de la autocorrelación.
%   Y=pitch_xcorr(X,FS,FI) obtiene el tono fundamental (pitch), de la
señal
%   X, con frecuencia de muestreo FS, y factor de interpolacion FI.
%   FI, es el factor de interpolacion de la función de autocorrelación
con
%   objeto de mejor la resulucion frecuencial.
% ------------------------------------------------------------------
%autocorrelación
c=correlacion_cruzada(x);
i=1;
%eliminamos el lobulo principal
while i+1<length(c) && (c(i+1)>0)
    c(i)=0;
    i=i+1;
end

[~,b]=max(c);
%interpolacion cubica para mejorar la precision
if b+1<length(c)
    rango=interp1(c(b-1:b+1),[1:(1/FI):3],'cubic');
    [~,incremento]=max(rango);
    incremento=incremento/FI;
else
    incremento=0;
end
y= 1/((b-2+incremento)*(1/fs));

function
[estado,energ_act]=det_estado(x,energ_max,umb_zcr,umb_energ,...
    umb_derivada,estado_anterior,signal)
%------------------------------------------------------------------
-----
% det_estado    Detecta el estado del tramo de audio.
%
[ESTADO,ENERG_ACT]=det_estado(X,ENERG_MAX,UMB_ZCR,UMB_ENERG,UMB_DERIVA
DA,ESTADO_ANTERIOR,SIGNAL)
%   ESTADO: variable de salida correspondiente al estado del tramo X,
%           'inicio': estado correspondiente al inicio del evento.
%           'fin': estado correspondiente al final del evento.
%           'ruido': estado de ruido.
%           'sostenido': estado intermedio entre el inicio del evento
y el
%           final del mismo
%   ENERG_ACT:variable de salida que almacena la energia maxima de la
señal X
%   X: variable de entrada. Señal bajo análisis.
%   ENERG_MAX: maxima energia registrada a lo largo de la toma de
ventanas.
%   UMB_ZCR: variable de entrada que define el umbral de tasa de
cruces por
%   cero entre el ruido y los tramos con señal.
%   UMB_ENERG: variable de entrada, que determina el umbral bajo el
cual se
%   considera ruido.
%   UMB_DERIVADA: Variable de entrada que establece el umbral que debe
%   superar la derivada del tramo X para poder considerarse inicio de
%   evento.
%   ESTADO_ANTERIOR: variable de entrada que se corresponde al estado
en
%   el que se encontraba la anterior ventana.
```

```
%   SIGNAL: cadena de caracteres que describe el tipo de señal que se
esta
%   analizando.
%       'voz': señal de voz
%       'silbido':señal compuesta a base de silbidos.
% ------------------------------------------------------------------
L=length(x);
energ_act=sum(x.^2)/L;%densidad de potencia

if strcmp(signal,'voz')
    switch estado_anterior
        case 'inicio'
            %si  no es ruido y supera el limite de energia
            if mean(zcr(x,40,20))<umb_zcr && energ_act>0.2*energ_max
                estado='sostenido';
            else
                estado='off';
            end

        case 'sostenido'
            %si  no es ruido y supera el limite de energia
            if mean(zcr(x,40,20))<umb_zcr && energ_act>0.2*energ_max
                estado='sostenido';
            else
                if energ_act<0.1*energ_max
                    estado='off';
                else
                    estado='sostenido';
                end
            end



        case 'off' %si vengo de un off
            if mean(zcr(x,40,20))<umb_zcr &&
energ_act>0.2*energ_max...
                    && max(diff(x))>umb_derivada
                estado='inicio';
            else
                estado='ruido';
            end

        case 'ruido'
            if mean(zcr(x,40,20))<umb_zcr && energ_act>umb_energ...
                    && max(diff(x))>umb_derivada
                estado='inicio';
            else
                estado='ruido';
            end
    end
else
    switch estado_anterior
        case 'inicio'
            %si  no es ruido y supera el limite de energia
            if energ_act>umb_energ
                estado='sostenido';
            else
                estado='off';
            end

        case 'sostenido'
```

```
            %si  no es ruido y supera el limite de energia
            if energ_act>umb_energ*1.2
                estado='sostenido';
            else
                if energ_act<umb_energ
                    estado='off';
                else
                    estado='sostenido';
                end
            end


        case 'off' %si vengo de un off
            if energ_act>umb_energ
                estado='inicio';
            else
                estado='ruido';
            end

        case 'ruido'
            if energ_act>umb_energ
                estado='inicio';
            else
                estado='ruido';
            end
    end
end

function Y = zcr(x,ventana, avance)
%-------------------------------------------------------------------
%ZCR    Tasa de cruces por cero.
%   Y=zcr(X,VENTANA,AVANCE), Almacena en el array Y, los valores de la
tasa
%   de cruces por cero existentes en la señal X, tomados en ventanas
de
%   longitud VANTANA, y avance AVANCE.
%   NOTA: AVANCE debe ser menor que VENTANA, se recomienda
solapamiento del
%   50% (AVANCE la mitad de VENTANA) o mayor.
%-------------------------------------------------------------------

x= normaliza(x);
curPos = 1;
L = length(x);
num_tramos = floor((L-ventana)/avance) + 1;
Y = zeros(num_tramos,1);
for i=1:num_tramos
    tramo = (x(curPos:curPos+ventana-1));
    tramo_sig = zeros(size(tramo));
    tramo_sig(2:end) = tramo(1:end-1);
    Y(i) = (1/(2*ventana)) *...
        sum(abs(sign(tramo)-sign(tramo_sig)));
    curPos = curPos + avance;
end
function y=get_vel(energia,umb_min,energ_max)
%-------------------------------------------------------------------
-----
%GET_VEL    Obtiene el valor del mensaje midi Velocity.
%   Y=get_vel(ENERGIA,UMB_MIN,ENERG_MAX) obtiene el valor de la
velocity
```

```
%   correspondiente a la diferencia entre la energia ENERGIA y la
energia
%   minima UMB_MIN, teniendo como máximo valor de la velocidad (127)
la
%   energia definida por ENERG_MAX
% ------------------------------------------------------------------
-----
min_vel=50;

y=round(((127-min_vel)/(energ_max-umb_min))*(energia-
umb_min)+min_vel);
if y>127
    y=127;
end
if y<0
    y=0;
end
function y=interpola_booleano(x,Lf)
%------------------------------------------------------------------
%INTERPOLA_BOOLEANO    Interpolacion de orden cero
%  Y=interpola_booleano(X,LF), interpola con orden cero la señal X,
%  resultando finalmente la señal Y interpolada con longitud LF
% ------------------------------------------------------------------

Li=length(x);
y=zeros(1,Lf);

%diferencia entre las longitudes
dif=Lf-Li;
%proporcion de muestras a interpolar entre cada muestra de entrada
m=dif/Li; %el numero de muestras a interpolar entre cada muestra de
entrada
incremento_m=m;
j=1;
i=1;
while i<=length(x)
    if m>=1 %si es mayor que 1 interpola
        y(j)=0;
        j=j+1;
        deci=m-fix(m);%nos quedamos con los decimales
        m=m-1;
        if fix(m)>=1
            inicio=j;
            while j<=inicio+fix(m) && fix(m)>0 %bucle por si acaso
supera uno y hay que interpolar mas de una
                y(j)=0;
                j=j+1;
                m=m-1;
            end
        else
            m=deci;
        end
    else
        %si no interpolo meto la entrada en la salida
        y(j)=x(i);
        j=j+1;
        m=m+incremento_m;
        i=i+1;
    end
end
function y=filtros(x,fs)
```

```
% ------------------------------------------------------------------
%FILTROS    banco de filtros.
%   Y=filtros(X,FS) filtra la señal X con frecuencia de muestreo FS,
con un
%   banco de 6 filtros:
%   Filtro paso bajo con frecuencia de corte 127 Hz
%   Filtro paso banda con frecuencias de corte 127 a 254 Hz
%   Filtro paso banda con frecuencias de corte 254 a 508 Hz
%   Filtro paso banda con frecuencias de corte 508 a 1016 Hz
%   Filtro paso banda con frecuencias de corte 1016 a 2032 Hz
%   Filtro paso banda con frecuencias de corte 2032 a 4064 Hz
%
%   Y: matriz  cuyas filas representan el resultado del filtrado de
cada
%   banda.
% ------------------------------------------------------------------

%---------FILTRO PASO BAJO------
fc=127;
wn=fc/fs;
N=20;%orden
%obtenemos coeficientes
B=fir1(N,wn);
%filtramos
x1=conv(B,x);
y=zeros(6,length(x1));
y(1,:)=x1;

%---------FILTRO PASO BANDA------
fc=[127 254];
wn=fc/fs;
%obtenemos coeficientes
B=fir1(N,wn,'DC-1');
%filtramos
x2=conv(B,x);
y(2,:)=x2;

%---------FILTRO PASO BANDA------
fc=[254 508];
wn=fc/fs;
%obtenemos coeficientes
B=fir1(N,wn,'DC-1');
%filtramos
x3=conv(B,x);
y(3,:)=x3;

%---------FILTRO PASO BANDA------
fc=[508 1016];
wn=fc/fs;
%obtenemos coeficientes
B=fir1(N,wn,'DC-1');
%filtramos
x4=conv(B,x);
y(4,:)=x4;

%---------FILTRO PASO BANDA------
fc=[1016 2032];
wn=fc/fs;
%obtenemos coeficientes
B=fir1(N,wn,'DC-1');
```

```
%filtramos
x5=conv(B,x);
y(5,:)=x5;



%---------FILTRO PASO BANDA------
fc=[2032 4064];
wn=fc/fs;
%obtenemos coeficientes
B=fir1(N,wn,'DC-1');
%filtramos
x6=conv(B,x);
y(6,:)=x6;
y(6,:)=zeros(1,length(x6));

function inicio_fin=detector_inicio_fin(x,fs,N,umb_ruido)
%-----------------------------------------------------------------------
-----
%DETECTOR_INICIO_FIN  obtiene los inicios y finales de notas de audio.
%   INICIO_FIN=detector_inicio_fin(X,FS,N,UMB_RUIDO), devuelve en el
array
%   INICIO_FIN las posiciones de los inicios y fin de eventos de la
señal
%   X, con frecuencia de muestreo FS.
%   La funcion toma como datos de entrada, el numero de puntos para
hacer
%   la envolvente N, y el umbral de ruido UMB_RUIDO
%
%   INICIO_FIN: cell array de una columna, que almacena en cada fila
los
%   datos de un evento. El primer elemento correspone al inicio, y el
%   segundo al final.
%-----------------------------------------------------------------------
-----
M=40;%numero de muestras para el enventanado para obtener la derivada
relativa


%---INICIALIZAR VARIABLES-------
xf=filtros(x,fs);

%se elige un tamaño de coger dos muestras por cada N muestras
[R,S]=size(xf);
% env=zeros(R,round(S/(N/2))-1);
% der_rel=zeros(R,round(S/(N/2))-1);
candidatos=zeros(R,round(S/(N/2))-1);
maximos=zeros(1,round(S/(N/2))-1);

env_original=envelope_reducida(x,N);




%-----CALCULO de la DERIVADA RELATIVA de cada banda
for l=1:R
    env(l,:)=envelope_reducida(xf(l,:),N); %calculo de la envolvente
se diezma por factor N/2
    %     env(l,:)=interp(env_red(l,:),N/2);
```

```
    env(l,:)=normaliza(env(l,:));

    k=1;
    for i=round(1:M:length(env)-M)%pensar forma de tomar tambien las
ultimas M muestras
        ventana=env(l,i:i+M-1);
        derivada=diff(ventana);
        derivada=[derivada,0];

        for j=1:length(derivada)
            if ventana(j)~=0
                der_rel(l,k)=derivada(j)/ventana(j);
            else
                der_rel(l,k)=0;
            end
            k=k+1;
        end
    end
    if i+M-1~=length(env)
        ventana=env(l,i+M-1:length(env)-1);
        derivada=diff(ventana);
        derivada=[derivada,0];

        for j=1:length(derivada)
            if ventana(j)~=0
                der_rel(l,k)=derivada(j)/ventana(j);
            else
                der_rel(l,k)=0;
            end
            k=k+1;
        end
    end
end

%     der_rel(l,:)=compand(der_rel(l,:),100,5,'mu/compressor');

C=length(der_rel(1,:));
umb_p=1.2;%umbral positivo para considerarlo candidato de INICIO de
nota.
umb_n=-0.3;%umbral negativo para considerarlo candidato de FIn de nota
for i=1:R
    for j=1:C
        if der_rel(i,j)>umb_n && der_rel(i,j)<umb_p
            der_rel(i,j)=0;
        end
    end
end

%
derivada_suma=der_rel(1,:);
for i=2:R
    derivada_suma=der_rel(i,:)+derivada_suma;
end

existe_temporal=interpola_booleano(derivada_suma,length(x));


% Se obtienes los inicios y fin de nota
inicio_fin=cell(1,1);
aux=[0 0]; %1 valor, 2 posicion
```

```
i=1;
j=1;
while i<=length(existe_temporal)
    if existe_temporal(i)~=0
        if existe_temporal(i)>0 %estamos buscando inicio de nota
            while i<=length(existe_temporal) && existe_temporal(i)>=0
                if aux(1)<existe_temporal(i)
                    aux(1)=existe_temporal(i);
                    aux(2)=i;
                end
                i=i+1;
            end
            i=i-1;
            inicio_fin{j,1}(1)=aux(2);

        else if ~isempty(inicio_fin{j,1})
                while i<=length(existe_temporal) &&
existe_temporal(i)<=0

                    if aux(1)>existe_temporal(i)
                        aux(1)=existe_temporal(i);
                        aux(2)=i;

                    end
                    i=i+1;
                end
                i=i-1;
                inicio_fin{j,1}(2)=aux(2);
                %se comprueba si se ha escogido un tramo correcto qeu
no es
                %ruido
                if inicio_fin{j,1}(1)==0
                    inicio_fin{j,1}(1)=1;
                end
                if  (i-inicio_fin{j,1}(1))<0.06/(1/fs) ||
max(x(inicio_fin{j,1}(1):inicio_fin{j,1}(2)))^2<=umb_ruido
                    inicio_fin{j,1}=[];
                else
                    j=j+1;
                end
            end
        end
    end
    i=i+1;
end
```

```
function PBU=pitch_bend(f_ant,f_act,bend_ant,vel_bend)
%------------------------------------------------------------------------
----
% pitch_bend    Obtiene valor de evento pitch bend (rueda de
modulacion)
%   PBU=pitch_bend(F_ANT,F_ACT,BEND_ANT,VEL_BEND) almacena en la
variable
%   PBU el valor del mensaje midi correspondiente a la variacion en
%   frecuencia de F_ANT a F_ACT, teniendo en cuenta el valor del pitch
bend
%   de la anterior ventana BEND_ANT, con una velocidad de cambio de
%   VEL_BEND.
%
%   PBU: variable de salida con el valor del mensaje [-8192 a 8192]
%   F_ANT: frecuencia de la anterior ventana tomada
%   F_ACT: frecuencia de la ventana actual.
%   BEND_ANT: valor del pitch bend anterior.
%   VEL_BEND: valor que define el incremento de cambio del pitch bend.
Para
%   cada sintetizador es diferente la forma en que la rueda de
modulacion
%   modifica el tono.
%------------------------------------------------------------------------
----
%tipico vel_bend=4;
%vel_bend=1/vel_bend;
%4 cents=1 PBU (unidad de pitch bend)
if f_act>f_ant
    cents=(log(f_act-f_ant))/(log(2^(1/1200)));
    PBU=round(cents/vel_bend);
    PBU=round(PBU*64/8192)+bend_ant;
else if f_act<f_ant
        cents=(log(f_ant-f_act))/(log(2^(1/1200)));
        cents=-cents;
        PBU=round(cents/vel_bend);
        PBU=round(PBU*64/8192)+bend_ant;
    else
        PBU=bend_ant;
    end
end


if PBU>127 && PBU<Inf
    PBU=127;
end
if PBU<0
    PBU=0;
end
if PBU == Inf
    PBU=64;
end
```

```
function [numero_nota,letra_nota,octava,desafinacion,f]=
frec2midi(f,fs)
% ------------------------------------------------------------------
----
%FREC2MIDI    Extraccion de parametros midi
%   [NUMERO_NOTA,LETRA_NOTA,OCTAVA,DESAFINACION,F]=frec2midi(F,FS)
obtiene
%   de la frecuencia F, el numero de nota en midi NUMERO_NOTA, su
%   correspondiente letra en notacion americana LETRA_NOTA, y el
numero de
%   la octava a la que pertenece OCTAVA. Además, tiene como parametro
de
%   salida diferencia en frecuencia entre la frecuencia de entrada F y
la
%   perfecta afinacion de la nota midi obtenida.
% ------------------------------------------------------------------
----

%----obtenemos el evento MIDI----
numero_nota=round(69+12*log2(f/440));
m=round(numero_nota(:));
o=floor(m/12)-1;
m=m-12*o+6*sign(f(:))-5;
a=('CDDEEFGGAABBCCDDEFFGGAAB')';
b=(' - -  - - -  # #  # # # ')';
letra_nota=setstr([a(m) mod(o,10)+'0' b(m)]);
%----en su correspondiente octava---
octava=floor(numero_nota/12)-1;



%----Obtenemos la desviacion frente a la nota MIDI---
f_midi= 440 * exp ((numero_nota-69) * log(2)/12); %la frecuencia de la
nota midi
desafinacion=f_midi-f;

if desafinacion>0     %si desafinacion positiva, comparamos con la
precision del algoritmo de
    nm=round(fs/f);                  %deteccion  entre la frecuencia
de la nota midi, y la siguiente posibilidad
    precision=(fs/nm)-(fs/(nm+1));
    if abs(precision)>abs(desafinacion)
        desafinacion=0;
    end
else %sino, la comparamos con la precision con respecto a la anterior
muestra
    nm=round(fs/f);                  %deteccion  entre la frecuencia
de la nota midi, y la siguiente posibilidad
    precision=(fs/nm)-(fs/(nm-1));
    if abs(precision)>abs(desafinacion)
        desafinacion=0;
    end
end

if length(letra_nota)==3
    letra_nota=[letra_nota(1),letra_nota(3),letra_nota(2)];
end
```

```
function [nota1,inc_oct_nota1,nota2,inc_oct_nota2]=
acorde(nota0,tipo_acorde)
%-------------------------------------------------------------------------
-----
%ACORDE    Calculo de acorde de triada
%
[NOTA1,INC_OCT_NOTA1,NOTA2_INC_OCT_NOTA2]=acorde(NOTA0,TIPO_ACORDE)
%   almacena en NOTA1 y NOTA2, las notas en notacion americana del
acorde
%   correspondiente a TIPO_ACORDE, con nota fundamental NOTA0. Como
%   parametro de salida se da el incremento de octava de cada nota si
esta
%   salta de una octava a otra (INC_OCT_NOTA1, INC_OCT_NOTA2).
%
%   Los tipos de acorde pueden ser:
%       o 'major'
%       o 'minor'
%       o 'power_chord'
%-------------------------------------------------------------------------
-----
inc_oct_nota1=0;
inc_oct_nota2=0;
letras=cell(1,12);
letras{1}='C';letras{2}='C#';
letras{3}='D';letras{4}='D#';
letras{5}='E';
letras{6}='F';letras{7}='F#';
letras{8}='G';letras{9}='G#';
letras{10}='A';letras{11}='A#';
letras{12}='B';

%buscamos la posicion de nuestra nota fundamental
i=1;
while i<=12 && ~strcmp(nota0(1),letras{i})
    i=i+1;
end

%i ahora es la posicion de la nota que tenemos
switch (tipo_acorde)
    case 'major'
        if i+4>12
            inc_oct_nota1=1;
            nota1=letras{i+4-12};
        else
            nota1=letras{i+4};
        end

        if i+7>12
            inc_oct_nota2=1;
            nota2=letras{i+7-12};
        else
            nota2=letras{i+7};
        end



    case 'minor'
        if i+3>12
            inc_oct_nota1=1;
            nota1=letras{i+3-12};
```

```
        else
            nota1=letras{i+3};
        end

        if i+7>12
            inc_oct_nota2=1;
            nota2=letras{i+7-12};
        else
            nota2=letras{i+7};
        end

    case 'power_chord'
        if i+5>12
            inc_oct_nota1=1;
            nota1=letras{i+5-12};
        else
            nota1=letras{i+5};
        end

        inc_oct_nota2=1;
        nota2=nota0(1);

end
```

Transcripcion rítmica

```
function varargout = percusion_nn_GUI(varargin)
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @percusion_nn_GUI_OpeningFcn, ...
    'gui_OutputFcn',  @percusion_nn_GUI_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before percusion_nn_GUI is made visible.
function percusion_nn_GUI_OpeningFcn(hObject, eventdata, handles,
varargin)
% Choose default command line output for percusion_nn_GUI
handles.output = hObject;
set(hObject,'name','TRANSCRIPCIÓN RÍTMICA');
% Update handles structure
guidata(hObject, handles);

imshow('Drum_kit.jpg')
warning off;

%inicializamos variables
assignin('base','entrenado_1',0);assignin('base','instrumento_1',35);
assignin('base','entrenado_2',0);assignin('base','instrumento_2',35);
assignin('base','entrenado_3',0);assignin('base','instrumento_3',35);
assignin('base','entrenado_4',0);assignin('base','instrumento_4',35);
assignin('base','escritura_habilitada',0);
assignin('base','real_time',0);
assignin('base','Elman',0);
assignin('base','FF',0);
assignin('base','optimizar',0);
assignin('base','minimo_velocity',30);
assignin('base','fichero_path_nombre','eventos_trans_ritmica.txt');
assignin('base','recalcular',0);
assignin('base','max_muestras_FF',1500);
assignin('base','transitorio',1);
assignin('base','umbral_deteccion',0.08)
assignin('base','bpm','60');
assignin('base','instrumento_1',35);
assignin('base','instrumento_2',38);
assignin('base','instrumento_2',42);
assignin('base','instrumento_2',49);


%Inicializamos los botones
set(handles.checkbox1,'Value',0);
```

```
set(handles.checkbox2,'Value',0);
set(handles.checkbox3,'Value',0);
set(handles.checkbox5,'Value',0);
set(handles.checkbox1,'Enable','off');
set(handles.checkbox2,'Enable','off');
set(handles.checkbox3,'Enable','off');
set(handles.checkbox5,'Enable','off');
set(handles.entrenar,'Enable','off');
set(handles.Escribir,'Value',0);
set(handles.real_time,'Value',0,...
    'Enable','on');
set(handles.buscar_fichero,'Enable','off');
set(handles.optimizar,'Enable','off');
set(handles.optimizar,'Value',0);
set(handles.Calidad,'Enable','off');
set(handles.baja,'Enable','off');
set(handles.alta,'Enable','off');
set(handles.FF,'Value',0);
set(handles.Elman,'Value',0);
set(handles.STOP,'Enable','off');
set(handles.pushbutton7,'Enable','on');
set(handles.FF,'Enable','off');
set(handles.Elman,'Enable','off');
set(handles.dinamica,'Value',30);
set(handles.nota1,'String','B1');
set(handles.nota2,'String','D2');
set(handles.nota3,'String','F#2');
set(handles.nota4,'String','C#3');
set(handles.umbral_deteccion,'Visible','off');
set(handles.recalcular,'Enable','off');
set(handles.bpm,'String','60');
set(handles.umbral_deteccion,'Value',0.08);


%Inicializamos los intrumentos
set(handles.popupmenu1,'Value',1);
set(handles.popupmenu5,'Value',4);
set(handles.popupmenu6,'Value',8);
set(handles.popupmenu7,'Value',15);
% UIWAIT makes percusion_nn_GUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = percusion_nn_GUI_OutputFcn(hObject, eventdata,
handles)
varargout{1} = handles.output;


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
instrumento=get(handles.popupmenu1,'Value');
assignin('base','instrumento_1',instrumento+34);
nota=instrumento2nota(instrumento+34);
set(handles.nota1,'String',nota);


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu3.
function popupmenu3_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function popupmenu3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu4.
function popupmenu4_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function popupmenu4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
get_inputs()
set(handles.checkbox1,'Enable','on');

% --- Executes on selection change in popupmenu5.
function popupmenu5_Callback(hObject, eventdata, handles)
instrumento=get(handles.popupmenu5,'Value');
assignin('base','instrumento_2',instrumento+34);
nota=instrumento2nota(instrumento+34);
set(handles.nota2,'String',nota);

% --- Executes during object creation, after setting all properties.
function popupmenu5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
real_time=evalin('base','real_time');
escribir=evalin('base','escritura_habilitada');
if ~real_time && ~escribir
    msgbox('Por favor, elija una salida para los datos MIDI','AYUDA');
end
if real_time
    set(handles.STOP,'Enable','on');
    set(handles.pushbutton7,'Enable','off');
    FF=evalin('base','FF');
    if FF
        net=evalin('base','net');
    else
        net_elm=evalin('base','net_elm');
```

```
    end
    coger_1=evalin('base','entrenado_1');
    coger_2=evalin('base','entrenado_2');
    coger_3=evalin('base','entrenado_3');
    coger_4=evalin('base','entrenado_4');
    ins_2=0;
    ins_1=0;
    ins_3=0;
    ins_4=0;
    velocity=cell(1,1);
    if coger_1==1
        ins_1=evalin('base','instrumento_1');
        velocity{1,1}=evalin('base','velocity_media_1');
    end
    if coger_2==1
        ins_2=evalin('base','instrumento_2');
        velocity{1,2}=evalin('base','velocity_media_2');
    end
    if coger_3==1
        ins_3=evalin('base','instrumento_3');
        velocity{1,3}=evalin('base','velocity_media_3');
    end
    if coger_4
        ins_4=evalin('base','instrumento_4');
        velocity{1,4}=evalin('base','velocity_media_4');
    end
    umbral_energia=evalin('base','umbral_energia');
    if evalin('base','FF')
        feat_elegidas=evalin('base','feat_elegidas');
    else
        feat_elegidas=1;
    end



    transitorio=evalin('base','transitorio');
    %VElocity de MIDI
    datos.minimo_velocity=evalin('base','minimo_velocity');
    velocity=cell2mat(velocity);

    %parametros de captacion de datos.
    datos.ventana=300;
    datos.sig=datos.ventana;
    datos.fs=8000;
    datos.transitorio=transitorio;

    %deteccion de estado
    datos.umbral_energ=umbral_energia;
    datos.umbral_derivada=0.005;
    datos.estado='ruido';
    datos.estado_anterior='ruido';
    datos.energ_max=0; %almacena el maximo valor para luego detectar
la caida.
    datos.dur_golpe=0.032; %(s)
    %parametros MIDI
    datos.channel=9; %el definido para la percusion.
    datos.velocity=velocity;
    datos.real_time=real_time;

    %Caracteristicas a escoger
    datos.feat_elegidas=feat_elegidas;
```

```
    %indices
    datos.ind_inputs=1;
    datos.contador=1;
    datos.signal=cell(1);
    datos.max_muestras_FF=evalin('base','max_muestras_FF');

    %red neuronal
    datos.FF=FF;
    if FF
        datos.net=net;
        datos.caracteristicas=zeros(2,1);
    else
        %red de elman
        datos.net_elm=net_elm;
        datos.numero_muestras_elman=40;
    end

    %escritura de fichero
    datos.escribir=escribir;
     datos.inputs_elman=cell(1,1);

    %abrimos el fichero de escritura midi si procede
    if datos.escribir==1
        nombre_fichero=evalin('base','fichero_path_nombre');
        datos.nombre=nombre_fichero;
        BPM=evalin('base','bpm');
        datos.file = fopen(datos.nombre,'w');
        fprintf(datos.file,['MThd | # of Tracks=1 | '...
            'Division=1000 \n\n Track #1 \n    | Tempo | BPM=%s
\n'],BPM);
    else
        nombre_fichero=0;
    end



    %MIDI
    datos.instrumentos=[ins_1, ins_2, ins_3, ins_4];
    import javax.sound.midi.*;
    datos.devices = MidiSystem.getMidiDeviceInfo;
    datos.mens = ShortMessage;

    datos.midiout = MidiSystem.getMidiDevice(datos.devices(2));
    datos.midiout.open;

    datos.midi.recep = datos.midiout.getReceiver;



    %programa elegido:

datos.mens.setMessage(ShortMessage.PROGRAM_CHANGE,1,datos.channel);
    datos.midi.recep.send(datos.mens, -1);

    %abrir la entrada targeta de sonido e inicializarla
    ai=analoginput('winsound');
    addchannel(ai,1);

    set(ai,...
        'SampleRate',datos.fs,...
```

```
        'SamplesPerTrigger',datos.sig,...
        'SamplesAcquiredFcnCount', datos.sig,...
        'TriggerRepeat', Inf,...
        'StopFcn', @cerrar_fichero,...
        'SamplesAcquiredFcn', @analisis_reproduccion_percusion ...
        );
    set (ai,'UserData',datos);
    handle_ai=guidata(hObject);
    handle_ai.ai=ai;
    guidata(hObject, handle_ai);
    start(ai);
    wait(ai,240);
    delete(ai),clear ai
end
if escribir && ~real_time
    set(handles.STOP,'Enable','on');
    set(handles.pushbutton7,'Enable','off');
    r = audiorecorder(8000, 16, 1);
    set(r,'Tag','recording');
    record(r)
    handle_r=guidata(hObject);
    handle_r.r=r;
    guidata(hObject, handle_r);
end


% --- Executes on button press in STOP.
function STOP_Callback(hObject, eventdata, handles)
real_time=evalin('base','real_time');
escribir=evalin('base','escritura_habilitada');
if real_time
    stop(handles.ai);
    delete (handles.ai), clear handles.ai
    set(handles.pushbutton7,'Enable','on');
    set(handles.STOP,'Enable','off');
end

if escribir && ~real_time
    set(handles.pushbutton7,'Enable','on');
    set(handles.STOP,'Enable','off');
    %Abrir fichero de escritura
    nombre_fichero=evalin('base','fichero_path_nombre');
    BPM=evalin('base','bpm');
    file = fopen(nombre_fichero,'w');
    fprintf(file,['MThd | # of Tracks=1 | '...
        'Division=1000 \n\n Track #1 \n       | Tempo | BPM=%s
\n'],BPM);

    %Obtener datos de audio
    stop(handles.r)
    grabacion=getaudiodata(handles.r);
    assignin('base','grabacion',grabacion);
    fs=get(handles.r,'SampleRate');

    %obtener datos del workspace

    energia_media(1)=evalin('base','velocity_media_1');
    energia_media(2)=evalin('base','velocity_media_2');
    energia_media(3)=evalin('base','velocity_media_3');
    energia_media(4)=evalin('base','velocity_media_4');
```

```
    instrumentos(1)=evalin('base','instrumento_1');
    instrumentos(2)=evalin('base','instrumento_2');
    instrumentos(3)=evalin('base','instrumento_3');
    instrumentos(4)=evalin('base','instrumento_4');

    recalcular=evalin('base','recalcular');
    if recalcular
        umbral_deteccion=evalin('base','umbral_deteccion');
    else
        umbral_deteccion=evalin('base','umbral_energia');
        umbral_deteccion=1*umbral_deteccion/max(energia_media);
    end

    feat_elegidas=evalin('base','feat_elegidas');
    numero_muestras_recurrente=40;
    numero_muestras_FF=evalin('base','max_muestras_FF');
    FF=evalin('base','FF');
    if FF
        net_FF=evalin('base','net');
        T=numero_muestras_FF;
    else
        net_recurrente=evalin('base','net_elm');
        T=numero_muestras_recurrente;
    end

    %--------analizar grabacion------
    %inicializacion de variables
    velocidad_minima=evalin('base','minimo_velocity');

    M=round(0.07/(1/fs)); %distancia minima entre eventos
    grabacion=grabacion';
    grabacion_normalizada=normaliza(abs(grabacion));
    E=length(grabacion);

    %----
    cortes=-1*ones(1,E);
    umbral_deteccion=evalin('base','umbral_deteccion');
    dur_golpe=0.032;
    energ_max=0;
    umbral_energ=evalin('base','umbral_energia');
    umbral_derivada=0.005;
    estado_anterior='ruido';
    inicio_anterior=1;
    min_dur=100;
    signal=cell(1,1);
    ind_inputs=1;
    contador=1;
    fs=8000;
    V=300; %tamaño de la ventana;
    for i=1:V:E-V
        tramo=grabacion(i:i+V);
        [estado,~]=estado_tomar_inputs(tramo,estado_anterior,...
            umbral_derivada,umbral_energ,energ_max);
        switch estado
            case 'inicio' %si es inicio, tomo la ventana copiandola en
el array de inputs
                if strcmp(estado_anterior,'inicio') %si no es la
primera ventana con señal
```

```
                    signal{ind_inputs,1}=[signal{ind_inputs},tramo];
                    M=max(abs(signal{ind_inputs,1}));
                    if M>energ_max
                        energ_max=M;
                    end
                else
                    signal{ind_inputs,1}=tramo;
                    energ_max=max(abs(signal{ind_inputs,1}));
                end

                [~,pos]=max(abs(tramo));
                inicio_evento=i+pos;

            case 'fin'
                if strcmp(estado_anterior,'inicio') && inicio_evento-
inicio_anterior>=min_dur
                    %limpiamos los inputs de ruido
                    i=4;
                    aux=zeros(1,fs*dur_golpe);
                    while i<length(signal{ind_inputs,1})
                        if max(abs(signal{ind_inputs,1}(i-3:i)))^2
>umbral_energ
                            if i+fs*dur_golpe<=
length(signal{ind_inputs,1})

aux=signal{ind_inputs,1}(i:i+fs*dur_golpe-1);
                                i=length(signal{ind_inputs,1});
                            else
                                aux=signal{ind_inputs,1}(i:end);
                                aux=[aux zeros(1,fs*dur_golpe-
length(aux))];
                                i=length(signal{ind_inputs,1});
                            end

                        end
                        i=i+1;
                    end

                    if isempty(aux)==0
                        if FF
                            input=get_features(aux,feat_elegidas);
                            [~,decision]=max(net_FF(input));
                        else

inputs_elman{1,ind_inputs}=normaliza(aux(1:numero_muestras_elman))';

[~,decision]=max(net_elm(normaliza(aux(transitorio:...
                                transitorio+numero_muestras_elman-
1)')));
                        end
                        velocity=fix(velocidad_minima+(127-
velocidad_minima)...
                            *(energ_max/max(abs(grabacion))));
                        cortes(inicio_evento)=velocity;
                        switch decision
                            case 1

[letra,octava]=num2letra(instrumentos(1));
                                escribe_fichero(file,'On
Note',inicio_evento/fs,letra,octava,velocity);
```

```
                                escribe_fichero(file,'Off
Note',(inicio_evento+V/2)/fs,letra,octava,100);

                            case 2

[letra,octava]=num2letra(instrumentos(2));
                                escribe_fichero(file,'On
Note',inicio_evento/fs,letra,octava,velocity);
                                escribe_fichero(file,'Off
Note',(inicio_evento+V/2)/fs,letra,octava,100);
                            case 3

[letra,octava]=num2letra(instrumentos(3));
                                escribe_fichero(file,'On
Note',inicio_evento/fs,letra,octava,velocity);
                                escribe_fichero(file,'Off
Note',(inicio_evento+V/2)/fs,letra,octava,100);
                            case 4

[letra,octava]=num2letra(instrumentos(4));
                                escribe_fichero(file,'On
Note',inicio_evento/fs,letra,octava,velocity);
                                escribe_fichero(file,'Off
Note',(inicio_evento+V/2)/fs,letra,octava,100);
                                energ_max=0;
                                signal={[]};
                                ind_inputs=ind_inputs+1;
                                inputs{1,ind_inputs}=[];

                        end
                    end
                    inicio_anterior=inicio_evento;
                end

            case 'ruido'
                energ_max=0;
                if strcmp(estado_anterior,'inicio')
                    ind_inputs=dind_inputs+1;

                end

        end
        estado_anterior=estado;
        contador=contador+1;
    end
    %actualizar graficos
    set(handles.recalcular,'Enable','on');
    %     set(handles.umbral_deteccion,'Visible','on');
    %actualizar el grafico de la GUI
    hold off;
    stem(cortes,'r');
    hold on;
    set(gca,'ylim',[0,140]);
    plot(abs(grabacion_normalizada)*100);
    linea_umbral=100*umbral_deteccion*ones(1,E);
    plot(linea_umbral,'g');
    grid;
    hold off;
    assignin('base','cortes',cortes);
```

```
    %cerrar fichero
    fprintf(file,'       | End of track |');
    fclose(file);
end


% --- Executes on selection change in popupmenu6.
function popupmenu6_Callback(hObject, eventdata, handles)
instrumento=get(handles.popupmenu6,'Value');
assignin('base','instrumento_3',instrumento+34);
nota=instrumento2nota(instrumento+34);
set(handles.nota3,'String',nota);


% --- Executes during object creation, after setting all properties.
function popupmenu6_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
get_inputs()
set(handles.checkbox2,'Enable','on');
inputs_2_elemento=evalin('base','inputs');

assignin('base','entrenado_2',1);
assignin('base','inputs_2',inputs_2_elemento);


% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
get_inputs()
set(handles.checkbox3,'Enable','on');
inputs_3_elemento=evalin('base','inputs');

assignin('base','entrenado_3',1);
assignin('base','inputs_3',inputs_3_elemento);

% --- Executes during object creation, after setting all properties.
function popupmenu7_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in popupmenu7.
function popupmenu7_Callback(hObject, eventdata, handles)
instrumento=get(handles.popupmenu7,'Value');
assignin('base','instrumento_4',instrumento+34);
nota=instrumento2nota(instrumento+34);
set(handles.nota4,'String',nota);

% --- Executes on button press in pushbutton10.
function pushbutton10_Callback(hObject, eventdata, handles)
get_inputs()
```

```
set(handles.checkbox5,'Enable','on');
inputs_4_elemento=evalin('base','inputs');

assignin('base','entrenado_4',1);
assignin('base','inputs_4',inputs_4_elemento);


% --- Executes on button press in checkbox5.
function checkbox5_Callback(hObject, eventdata, handles)
if (get(hObject,'Value') == get(hObject,'Max'))
    inputs_cuarto_elemento=evalin('base','inputs');

    assignin('base','entrenado_4',1);
    assignin('base','inputs_4',inputs_cuarto_elemento);

    inputs_cuarto_elemento_elm=evalin('base','inputs_elman');
    assignin('base','inputs_4_elm',inputs_cuarto_elemento_elm);

    velocity=evalin('base','velocity_media');
    assignin('base','velocity_media_4',velocity);

    ent_1=evalin('base','entrenado_1');
    ent_2=evalin('base','entrenado_2');
    ent_3=evalin('base','entrenado_3');
    ent_4=evalin('base','entrenado_4');
    if ent_1 && ent_2 && ent_3 && ent_4
        set(handles.FF,'Enable','on');
        set(handles.Elman,'Enable','on');
    end
else
    assignin('base','entrenado_4',0);
    % Checkbox is not checked-take appropriate action
end


function get_inputs()
%-------------------------------------------------------------------------
-----
%GET_INPUTS   Funcion que obtiene los golpes de una clase grabados a
traves
%de la entrada de microfono.
%-------------------------------------------------------------------------
-----
%parametros de captacion
datos.ventana=500;
datos.sig=datos.ventana;
datos.num_iteraciones=400;
datos.num_inputs=41;
datos.fs=8000;

%deteccion de estado
datos.estado='ruido';
datos.estado_anterior='ruido';
datos.energ_max=0; %almacena el maximo valor para luego detectar la
caida.

datos.ind_inputs=1;
datos.signal=cell(1,1);
```

```
datos.inputs=cell(1,1);

datos.inicio_evento=1;
datos.velocity=cell(1,1); % valores medios energia por golpe.
datos.dur_golpe=0.032; %32 milisegundos es la duracion que se va a
tomar
datos.inputs_elman=cell(1,1);
datos.numero_muestras_elman=256;
datos.max_muestras_FF=200;
assignin('base','max_muestras_FF',datos.max_muestras_FF);

%deteccion de golpeo
datos.umbral_derivada=0.005;
datos.n=cell(1);

%deteccion de ruido.
datos.vent_ruido=21; %numero de ventanas para detectar el ruido.
datos.ind_ruido=1;
datos.wb=waitbar(0,'Midiendo ruido...');

%abrir la entrada targeta de sonido e inicializarla
ai=analoginput('winsound');
addchannel(ai,1);

set(ai,...
    'SampleRate',datos.fs,...
    'SamplesPerTrigger',datos.sig,...
    'SamplesAcquiredFcnCount', datos.sig,...
    'TriggerRepeat', datos.num_iteraciones+datos.vent_ruido,...
    'StopFcn', @mandar_inputs, ...
    'SamplesAcquiredFcn', @tomar_ventanas ...
    );
set (ai,'UserData',datos);
start(ai);

clear ai,delete ai


function mandar_inputs(obj,event)
%-------------------------------------------------------------------
-----
%MANDAR_INPUTS    StopFcn correspondiente a la finalizacion de
get_inputs,
%ésta es la encargada de informar al usuario si la entrada ha sido
creada,
%y en ese caso, mandarla al workspace donde se procesaran por otras
%funciones.
%-------------------------------------------------------------------
-----
ai=obj;
datos=ai.UserData;

[~,C]=size(datos.inputs);
if C>=datos.num_inputs
    msgbox('Por favor, valídela en el menú de grabación','ENTRADA
CREADA')
else
    msgbox('No ha golpeado la superficie suficientes veces. Por favor,
vuelva a grabar' ...
        ,'ENTRADA NO CREADA');
```

```
end

assignin('base','inputs',datos.inputs);
assignin('base','inputs_elman',datos.inputs_elman);
assignin('base','umbral_energia',datos.umbral_energ);
assignin('base','velocity_media',mean(cell2mat(datos.velocity)));

function tomar_ventanas(obj,event)
%-------------------------------------------------------------------------
-----
%TOMAR_VENTANAS    Funcion recurrente que coge un tramo de señal y la
%procesa obteniendo las caracteristicas necesarias.
%-------------------------------------------------------------------------
-----

ai=obj;
datos=ai.UserData;

if datos.ind_inputs>=datos.num_inputs
    stop(ai);
end
datos_nuevos=getdata(ai); %cogemos datos


if datos.ind_ruido==datos.vent_ruido
    close(datos.wb)
    datos.message = msgbox('Golpee la superficie repetidas
veces','CREAR ENTRADAS','help');
    a=cell2mat(datos.n);
    datos.umbral_energ=max(a)*1.2;
    datos.ind_ruido=datos.ind_ruido+1;
end
if datos.ind_ruido>=datos.vent_ruido

[estado,energ_max]=estado_tomar_inputs(datos_nuevos,datos.estado_anter
ior,datos.umbral_derivada,datos.umbral_energ,datos.energ_max);
else
    estado='ruido';
end
switch estado
    case 'inicio' %si es inicio, tomo la ventana copiandola en el
array de inputs
        if exist('datos.wb','var')
            close (datos.wb);
        end

        if strcmp(datos.estado_anterior,'inicio') %si no es la primera
ventana con señal

datos.signal{datos.ind_inputs,1}=[datos.signal{datos.ind_inputs},datos
_nuevos'];
        else
            datos.signal{datos.ind_inputs,1}=datos_nuevos';
        end
        if energ_max>datos.energ_max
            datos.energ_max=energ_max;
        end


    case 'fin'
```

```
        datos.energ_max=0;
        datos.ind_ventanas=1;

        if strcmp(datos.estado_anterior,'inicio')
            %limpiamos los inputs de ruido
            i=4;
            aux=zeros(1,datos.fs*datos.dur_golpe);
            while i<length(datos.signal{datos.ind_inputs,1})
                if max(abs(datos.signal{datos.ind_inputs,1}(i-3:i)))^2
>datos.umbral_energ
                    if i+datos.fs*datos.dur_golpe<=
length(datos.signal{datos.ind_inputs,1})

aux=datos.signal{datos.ind_inputs,1}(i:i+datos.fs*datos.dur_golpe-1);
                        i=length(datos.signal{datos.ind_inputs,1});
                    else
                        aux=datos.signal{datos.ind_inputs,1}(i:end);
                        aux=[aux zeros(1,datos.fs*datos.dur_golpe-
length(aux))];
                        i=length(datos.signal{datos.ind_inputs,1});
                    end

                end
                i=i+1;
            end
            if isempty(aux)==0

[datos.inputs{1,datos.ind_inputs},~]=get_features(aux,[1:17]);

datos.inputs_elman{1,datos.ind_inputs}=normaliza(aux)';
                datos.velocity{1,datos.ind_inputs}=max(aux)^2;
                if isempty(datos.inputs)~=1
                    datos.signal={[]};
                    datos.ind_inputs=datos.ind_inputs+1;
                    datos.inputs{1,datos.ind_inputs}=[];
                end
            end
        end

    case 'ruido'
        datos.energ_max=0;
        if datos.ind_ruido<=datos.vent_ruido
            datos.n{datos.ind_ruido}=max(datos_nuevos.^2);

            waitbar(((datos.ind_ruido/(datos.vent_ruido-1))),...
                datos.wb,sprintf('ESPERE \n Midiendo ruido: %d%%
completado...'...
                ,round((datos.ind_ruido/datos.vent_ruido)*100+5)));
            if strcmp(datos.estado_anterior,'inicio')
                datos.ind_inputs=datos.ind_inputs+1;



            end
            datos.ind_ruido=datos.ind_ruido+1;
        end

end
disp(estado)
datos.estado_anterior=estado;
set(obj,'UserData',datos);
```

```
% --- Executes on button press in entrenar.
function entrenar_Callback(hObject, eventdata, handles)
% hObject    handle to entrenar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user datos (see GUIDATA)
set(handles.pushbutton7,'Enable','on');
FF=evalin('base','FF');
optimizar=evalin('base','optimizar');
Elman=evalin('base','Elman');


num_elem=0;
coger_1=evalin('base','entrenado_1');
if coger_1==1
    num_elem=num_elem+1;
end
coger_2=evalin('base','entrenado_2');
if coger_2==1
    num_elem=num_elem+1;
end
coger_3=evalin('base','entrenado_3');
if coger_3==1
    num_elem=num_elem+1;
end
coger_4=evalin('base','entrenado_4');
if coger_4
    num_elem=num_elem+1;
end

%Evaluamos que features son convenientes:
inputs_evaluacion=cell(1,4);

if coger_1==1
    inputs_1=evalin('base','inputs_1');
    inputs_1_elm=evalin('base','inputs_1_elm');
    inputs_1=inputs_1(:,1:end-1);


    targets=ones(num_elem,length(inputs_1));
    targets=targets.*(0);
    targets(1,:)=1;

    %targets Elman
    targets_elman=(0)*ones(num_elem,length(inputs_1));
    targets_elman(1,:)=1;
    %para evaluar
    inputs_evaluacion{1,1}=inputs_1;

end

if coger_2==1
    inputs_2=evalin('base','inputs_2');
    inputs_2_elm=evalin('base','inputs_2_elm');
    inputs_2=inputs_2(:,1:end-1);
    target_2=ones(num_elem,length(inputs_2));
    target_2=target_2.*(0);
    target_2(2,:)=1;
```

```
    targets=[targets,target_2];

    %para elman
    targets_2_elman=(0)*ones(num_elem,length(inputs_2));
    targets_2_elman(2,:)=1;
    targets_elman=[targets_elman,targets_2_elman];

    %para evaluar
    inputs_evaluacion{1,2}=inputs_2;

end

if coger_3==1
    inputs_3=evalin('base','inputs_3');
    inputs_3_elm=evalin('base','inputs_3_elm');
    inputs_3=inputs_3(:,1:end-1);

    target_3=ones(num_elem,length(inputs_3));
    target_3=target_3.*(0);
    target_3(3,:)=1;
    targets=[targets,target_3];

    %para elman
    targets_3_elman=(0)*ones(num_elem,length(inputs_3));
    targets_3_elman(3,:)=1;
    targets_elman=[targets_elman,targets_3_elman];

    %para evaluar
    inputs_evaluacion{1,3}=inputs_3;

end

if coger_4==1
    inputs_4=evalin('base','inputs_4');
    inputs_4_elm=evalin('base','inputs_4_elm');
    inputs_4=inputs_4(:,1:end-1);
    %inputs43=cell2mat(inputs_3);
    target_4=ones(num_elem,length(inputs_4));
    target_4=target_4.*(0);
    target_4(4,:)=1;
    targets=[targets,target_4];

    %para elman
    targets_4_elman=(0)*ones(num_elem,length(inputs_4));
    targets_4_elman(4,:)=1;
    targets_elman=[targets_elman,targets_4_elman];

    %para evaluar
    inputs_evaluacion{1,4}=inputs_4;

end


%SELECCION DE INPUTS FEEDFORWARD
feat_elegidas=eval_features(inputs_evaluacion);
numero_muestras_elman=40;

[NF,~]=size(feat_elegidas); %number of features
```

```
for i=1:NF
    if coger_4
        if i==1
            inputs_4=cell2mat(inputs_4);
            inputs_3=cell2mat(inputs_3);
            inputs_2=cell2mat(inputs_2);
            inputs_1=cell2mat(inputs_1);

            transitorio_general=1; %para evaluacion mejorar!!

            inputs_1_elm=cell2mat(inputs_1_elm);
            inputs_2_elm=cell2mat(inputs_2_elm);
            inputs_3_elm=cell2mat(inputs_3_elm);
            inputs_4_elm=cell2mat(inputs_4_elm);
            inputs_1_elm=normaliza_mat(inputs_1_elm...

(transitorio_general:transitorio_general+numero_muestras_elman-1,:));
            inputs_2_elm=normaliza_mat(inputs_2_elm...

(transitorio_general:transitorio_general+numero_muestras_elman-1,:));
            inputs_3_elm=normaliza_mat(inputs_3_elm...

(transitorio_general:transitorio_general+numero_muestras_elman-1,:));
            inputs_4_elm=normaliza_mat(inputs_4_elm...

(transitorio_general:transitorio_general+numero_muestras_elman-1,:));
            inputs_elman_final=[ inputs_1_elm, inputs_2_elm,
inputs_3_elm, inputs_4_elm];
        end
        inputs_final(i,:)=[inputs_1(feat_elegidas(i),:),...

inputs_2(feat_elegidas(i),:),inputs_3(feat_elegidas(i),:),inputs_4(fea
t_elegidas(i),:)];

    else if coger_3
            if i==1
                inputs_3=cell2mat(inputs_3);
                inputs_2=cell2mat(inputs_2);
                inputs_1=cell2mat(inputs_1);

                transitorio_general=1;

                inputs_1_elm=normaliza_mat(inputs_1_elm...

(transitorio_general:transitorio_general+numero_muestras_elman-1,:));
                inputs_2_elm=normaliza_mat(inputs_2_elm...

(transitorio_general:transitorio_general+numero_muestras_elman-1,:));
                inputs_3_elm=normaliza_mat(inputs_3_elm...

(transitorio_general:transitorio_general+numero_muestras_elman-1,:));

                inputs_elman_final=[ inputs_1_elm, inputs_2_elm,
inputs_3_elm];
            end
            inputs_final(i,:)=[inputs_1(feat_elegidas(i),:),...

inputs_2(feat_elegidas(i),:),inputs_3(feat_elegidas(i),:)];
        else if coger_2
```

```
                if i==1
                    inputs_2=cell2mat(inputs_2);
                    inputs_1=cell2mat(inputs_1);

                    transitorio_general=1;
                    inputs_1_elm=normaliza_mat(inputs_1_elm...
(transitorio_general:transitorio_general+numero_muestras_elman-1,:));
                    inputs_2_elm=normaliza_mat(inputs_2_elm...
(transitorio_general:transitorio_general+numero_muestras_elman-1,:));

                    inputs_elman_final=[ inputs_1_elm, inputs_2_elm];
                end
inputs_final(i,:)=[inputs_1(feat_elegidas(i),:),inputs_2(feat_elegidas
(i),:)];
            else if coger_1
                    if i==1
                        inputs_1=cell2mat(inputs_1);
                        transitorio_general=1;
assignin('base','transitorio',transitorio_general);
                        inputs_elman_final=inputs_1_elm;
                    end
                    inputs_final(i,:)=[inputs_1(feat_elegidas(i),:)];
                end
            end
        end
    end


end



inputs_todas=cell2mat(inputs_evaluacion{1,1});
inputs_todas=[inputs_todas,cell2mat(inputs_evaluacion{1,2})];
inputs_todas=[inputs_todas,cell2mat(inputs_evaluacion{1,3})];
inputs_todas=[inputs_todas,cell2mat(inputs_evaluacion{1,4})];
inputs_final=[];
feat_elegidas=sort(feat_elegidas);
for i=1:length(feat_elegidas)
    inputs_final(i,:)=inputs_todas(feat_elegidas(i),:);
end
%entrenar RED NEURONAL FEEDFORWARD

% creamos la red neuronal feedforward
if optimizar
    neuronas_primera_capa=14;
    neuronas_repeticion=evalin('base','neuronas_repeticion');
    neuronas=[7:neuronas_repeticion(1):30];
    h = waitbar(0,'Optimizando red neuronal...') ;
    total=length(neuronas)*neuronas_repeticion(2);
    c=1;
    for j=1:length(neuronas)
        for i=1:neuronas_repeticion(2)
            net =
patternnet([neuronas_primera_capa,neuronas(j)],'trainlm');
            net.trainParam.showWindow=0;
```

```
            net.layers{3}.transferFcn='purelin';
            net.layers{2}.transferFcn='tansig';
            net.layers{1}.transferFcn='logsig';

            [~,tr] = train(net,inputs_final,targets);
            evaluacion(i,j)=tr.best_vperf;
            waitbar(c/total);
            c=c+1;
        end

    end
    close (h);
    for j=1:length(evaluacion(1,:))
        media_evaluacion(j)=mean(evaluacion(:,j));
    end
    [~,red_elegida]=min(media_evaluacion);
    net =
patternnet([neuronas_primera_capa,neuronas(red_elegida)],'trainlm');
else
    net = patternnet([14,7],'trainlm');
end

if FF
    net.layers{3}.transferFcn='purelin';
    net.layers{2}.transferFcn='tansig';
    net.layers{1}.transferFcn='logsig';

    %-----Finalizacion de entrenamiento
    net.trainParam.min_grad=10^-10;
    net.trainParam.max_fail=14;
    net.trainParam.time=80;
    net.trainParam.epochs=500;

    [net,tr] = train(net,inputs_final,targets);

    assignin('base','net',net);
    assignin('base','feat_elegidas',feat_elegidas);
else
    net=0;
end


% creamos la red de elman

if Elman

    net_elm = layrecnet([0],[30],'trainlm');
    net_elm.layers{1}.transferFcn='tansig';
    net_elm.plotFcns=[net_elm.plotFcns,'plotconfusion'];
    net_elm.trainParam.showWindow=1;
    net_elm.layerWeights{1,1}.delays = 1;
    %-----Finalizacion de entrenamiento
    net_elm.trainParam.min_grad=10^-9;
    net_elm.trainParam.max_fail=8;
    net_elm.trainParam.time=80;

    [net_elm,tr_elm] =
train(net_elm,inputs_elman_final,targets_elman);
```

```
else
    net_elm=0;
end
assignin('base','net_elm',net_elm);


function cerrar_fichero (obj, event)
datos=obj.UserData;
if datos.escribir
    fprintf(datos.file,'       | End of track |');
    fclose(datos.file);
end



function analisis_reproduccion_percusion(obj,event)
%ANALISIS_REPRODUCCION_PERCUSION   Funcion recurrente que calcula las
%caracteristicas correspondientes a una ventana y mada los datos al
%sintetizador o al fichero de texto, como se haya especificado
import javax.sound.midi.*;

ai=obj;
datos=ai.UserData;
tic

datos_nuevos=getdata(ai); %cogemos datos

[estado,energ_max]=estado_tomar_inputs(datos_nuevos,datos.estado_anter
ior,...
    datos.umbral_derivada,datos.umbral_energ,datos.energ_max);
switch estado
    case 'inicio' %si es inicio, tomo la ventana copiandola en el
array de inputs
        if strcmp(datos.estado_anterior,'inicio') %si no es la primera
ventana con señal

datos.signal{datos.ind_inputs,1}=[datos.signal{datos.ind_inputs},datos
_nuevos'];

        else
            datos.signal{datos.ind_inputs,1}=datos_nuevos';
            [~,pos]=max(abs(datos_nuevos));

datos.inicio_evento=(pos+(datos.contador)*datos.sig)/datos.fs;
        end
        if energ_max>datos.energ_max
            datos.energ_max=energ_max;
            [~,pos]=max(abs((datos_nuevos)));

datos.inicio_evento=(pos+(datos.contador)*datos.sig)/datos.fs;
        end

    case 'fin'

        datos.energ_max=0;

        if strcmp(datos.estado_anterior,'inicio')
            %limpiamos los inputs de ruido
            i=4;
```

```
            aux=zeros(1,datos.fs*datos.dur_golpe);
            while i<length(datos.signal{datos.ind_inputs,1})
                if max(abs(datos.signal{datos.ind_inputs,1}(i-3:i)))^2
>datos.umbral_energ
                    if i+datos.fs*datos.dur_golpe<=
length(datos.signal{datos.ind_inputs,1})

aux=datos.signal{datos.ind_inputs,1}(i:i+datos.fs*datos.dur_golpe-1);
                        i=length(datos.signal{datos.ind_inputs,1});
                    else
                        aux=datos.signal{datos.ind_inputs,1}(i:end);
                        aux=[aux zeros(1,datos.fs*datos.dur_golpe-
length(aux))];
                        i=length(datos.signal{datos.ind_inputs,1});
                    end

                end
                i=i+1;
            end

            if isempty(aux)==0
                if datos.FF
                    datos.input=get_features(aux,datos.feat_elegidas);
                    [~,decision]=max(datos.net(datos.input));
                else

datos.inputs_elman{1,datos.ind_inputs}=normaliza...
                        (aux(1:datos.numero_muestras_elman))';
                    [~,decision]=max(datos.net_elm(normaliza...

(aux(datos.transitorio:datos.transitorio+datos.numero_muestras_elman-
1)')));
                end

                switch decision
                    case 1
                        %obtener velocity

velocity=get_velocity(datos.velocity(1,decision),...

max(abs(aux))^2,datos.umbral_energ,datos.minimo_velocity);
                        if datos.real_time

datos.mens.setMessage(ShortMessage.NOTE_ON,...
                                datos.channel, datos.instrumentos(1),
velocity);
                            datos.midi.recep.send(datos.mens, -1);
                        end




                        if datos.escribir

[letra,octava]=num2letra(datos.instrumentos(1));
                            escribe_fichero(datos.file,'On Note',...

datos.inicio_evento,letra,octava,velocity);
                            escribe_fichero(datos.file,'Off Note',...

(datos.contador+1)*datos.sig/datos.fs,letra,octava,120);
```

```
                        end

                    case 2
                        %obtener velocity

velocity=get_velocity(datos.velocity(1,decision),...

max(abs(aux))^2,datos.umbral_energ,datos.minimo_velocity);

                        if datos.real_time

datos.mens.setMessage(ShortMessage.NOTE_ON,...
                                datos.channel, datos.instrumentos(2),
velocity);
                            datos.midi.recep.send(datos.mens, -1);
                        end

                        if datos.escribir

[letra,octava]=num2letra(datos.instrumentos(2));
                            escribe_fichero(datos.file,'On Note',...

datos.inicio_evento,letra,octava,velocity);
                            escribe_fichero(datos.file,'Off Note',...

(datos.contador+1)*datos.sig/datos.fs,letra,octava,120);
                        end
                    case 3

velocity=get_velocity(datos.velocity(1,decision),...

max(abs(aux))^2,datos.umbral_energ,datos.minimo_velocity);
                        if datos.real_time

datos.mens.setMessage(ShortMessage.NOTE_ON,...
                                datos.channel, datos.instrumentos(3),
velocity);
                            datos.midi.recep.send(datos.mens, -1);
                        end

                        if datos.escribir

[letra,octava]=num2letra(datos.instrumentos(3));
                            escribe_fichero(datos.file,'On Note',...

datos.inicio_evento,letra,octava,velocity);
                            escribe_fichero(datos.file,'Off Note',...

(datos.contador+1)*datos.sig/datos.fs,letra,octava,120);
                        end

                    case 4

velocity=get_velocity(datos.velocity(1,decision),...

max(abs(aux))^2,datos.umbral_energ,datos.minimo_velocity);
                        if datos.real_time

datos.mens.setMessage(ShortMessage.NOTE_ON,...
```

```
                                datos.channel, datos.instrumentos(4),
velocity);
                            datos.midi.recep.send(datos.mens, -1);
                        end

                        if datos.escribir

[letra,octava]=num2letra(datos.instrumentos(4));
                            escribe_fichero(datos.file,'On Note',...

datos.inicio_evento,letra,octava,velocity);
                            escribe_fichero(datos.file,'Off Note',...

(datos.contador+1)*datos.sig/datos.fs,letra,octava,120);
                        end

                end
                datos.signal={[]};
                datos.ind_inputs=datos.ind_inputs+1;
                datos.inputs{1,datos.ind_inputs}=[];
            end
        end

    case 'ruido'
        datos.energ_max=0;
        if strcmp(datos.estado_anterior,'inicio')

            datos.ind_inputs=datos.ind_inputs+1;

        end


end
disp(estado)
datos.estado_anterior=estado;
datos.contador=datos.contador+1;
set(obj,'UserData',datos);


% --- Executes on button press in checkbox1.
function checkbox1_Callback(hObject, eventdata, handles)
if (get(hObject,'Value') == get(hObject,'Max'))
    inputs_primer_elemento=evalin('base','inputs');

    inputs_primer_elemento_elm=evalin('base','inputs_elman');
    assignin('base','inputs_1_elm',inputs_primer_elemento_elm);

    assignin('base','entrenado_1',1);
    assignin('base','inputs_1',inputs_primer_elemento);

    velocity=evalin('base','velocity_media');
    assignin('base','velocity_media_1',velocity);

else
    assignin('base','entrenado_1',0);
    % Checkbox is not checked-take appropriate action
end
```

```
% --- Executes on button press in checkbox2.
function checkbox2_Callback(hObject, eventdata, handles)
if (get(hObject,'Value') == get(hObject,'Max'))
    inputs_segundo_elemento=evalin('base','inputs');

    assignin('base','entrenado_2',1);
    assignin('base','inputs_2',inputs_segundo_elemento);

    inputs_segundo_elemento_elm=evalin('base','inputs_elman');
    assignin('base','inputs_2_elm',inputs_segundo_elemento_elm);

    velocity=evalin('base','velocity_media');
    assignin('base','velocity_media_2',velocity);
else
    assignin('base','entrenado_2',0);
    % Checkbox is not checked-take appropriate action
end

% --- Executes on button press in checkbox3.
function checkbox3_Callback(hObject, eventdata, handles)
if (get(hObject,'Value') == get(hObject,'Max'))
    inputs_tercer_elemento=evalin('base','inputs');

    assignin('base','entrenado_3',1);
    assignin('base','inputs_3',inputs_tercer_elemento);

    inputs_tercer_elemento_elm=evalin('base','inputs_elman');
    assignin('base','inputs_3_elm',inputs_tercer_elemento_elm);

    velocity=evalin('base','velocity_media');
    assignin('base','velocity_media_3',velocity);
else
    assignin('base','entrenado_3',0);
    % Checkbox is not checked-take appropriate action
end


% --- Executes on button press in Escribir.
function Escribir_Callback(hObject, eventdata, handles)
if (get(hObject,'Value') == get(hObject,'Max'))

    assignin('base','escritura_habilitada',1);
    set(handles.buscar_fichero,'Enable','on');
    set(handles.real_time,'Enable','off');
    set(handles.real_time,'Value',0);
else
    assignin('base','escritura_habilitada',0);
    set(handles.buscar_fichero,'Enable','off');
    set(handles.real_time,'Enable','on');
    % Checkbox is not checked-take appropriate action
end


% --- Executes on button press in buscar_fichero.
function buscar_fichero_Callback(hObject, eventdata, handles)
[nombre_fichero,path_fichero] = uiputfile({'*.txt','Archivo de texto
(*.txt)'},'Nuevo archivo');
```

```
assignin('base','fichero_path_nombre',[path_fichero nombre_fichero]);


% --- Executes on button press in real_time.
function real_time_Callback(hObject, eventdata, handles)
if (get(hObject,'Value') == get(hObject,'Max'))

    assignin('base','real_time',1);
    set(handles.Escribir,'Value',0,...
        'Enable','off');
else
    assignin('base','real_time',0);
    set(handles.Escribir,'Enable','on');
    % Checkbox is not checked-take appropriate action
end

% Hint: get(hObject,'Value') returns toggle state of real_time


% --- Executes on button press in optimizar.
function optimizar_Callback(hObject, eventdata, handles)
if (get(hObject,'Value') == get(hObject,'Max'))

    set(handles.Calidad,'Enable','on');
    set(handles.baja,'Enable','on');
    set(handles.alta,'Enable','on');
    assignin('base','optimizar',1);
else
    set(handles.Calidad,'Enable','off');
    set(handles.baja,'Enable','off');
    set(handles.alta,'Enable','off');
    assignin('base','optimizar',0);
    % Checkbox is not checked-take appropriate action
end

% Hint: get(hObject,'Value') returns toggle state of optimizar


% --- Executes on slider movement.
function Calidad_Callback(hObject, eventdata, handles)
calidad=get(handles.Calidad,'Value');
neu_rep=[0 0];%distancia entre neuronas y repeticiones de cada red

if calidad<1/8
    neu_rep=[1,14];
else if calidad>=1/8 && calidad <2/8
        neu_rep=[1,12];
    else if calidad>2/8 && calidad <3/8
            neu_rep=[2,10];
        else if calidad>3/8 && calidad <4/8
                neu_rep=[2,8];
            else if calidad>4/8 && calidad <5/8
                    neu_rep=[3,8];
                else if calidad>5/8 && calidad <6/8
                        neu_rep=[3,7];
                    else if calidad>6/8 && calidad <7/8
                            neu_rep=[4,7];
                        else if calidad>7/8 && calidad <=8/8
                                neu_rep=[4,6];
```

```
                            end
                        end
                    end
                end
            end
        end
    end
end

assignin('base','neuronas_repeticion',neu_rep);
% Hints: get(hObject,'Value') returns position of slider

%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider


% --- Executes during object creation, after setting all properties.
function Calidad_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in FF.
function FF_Callback(hObject, eventdata, handles)
if (get(hObject,'Value') == get(hObject,'Max'))

    assignin('base','FF',1);
    assignin('base','Elman',0);
    set(handles.optimizar,'Enable','on');
    set(handles.Elman,'Value',0);
    ent_1=evalin('base','entrenado_1');
    ent_2=evalin('base','entrenado_2');
    ent_3=evalin('base','entrenado_3');
    ent_4=evalin('base','entrenado_4');

    if ent_1 && ent_2 && ent_3 && ent_4
        set(handles.entrenar,'Enable','on');
    end

else
    assignin('base','FF',0);
    assignin('base','optimizar',0);

    set(handles.optimizar,'Enable','off');
    set(handles.optimizar,'Value',0);
    set(handles.baja,'Enable','off');
    set(handles.alta,'Enable','off');
    set(handles.entrenar,'Enable','off');


    % Checkbox is not checked-take appropriate action
end

% Hint: get(hObject,'Value') returns toggle state of FF
```

```
% --- Executes on button press in Elman.
function Elman_Callback(hObject, eventdata, handles)
if (get(hObject,'Value') == get(hObject,'Max'))
    assignin('base','Elman',1);
    assignin('base','FF',0);
    assignin('base','optimizar',0);
    set(handles.FF,'Value',0);
    %     set(handles.optimizar,'Value',0);
    set(handles.optimizar,'Enable','off');
    set(handles.Calidad,'Enable','off');

    ent_1=evalin('base','entrenado_1');
    ent_2=evalin('base','entrenado_2');
    ent_3=evalin('base','entrenado_3');
    ent_4=evalin('base','entrenado_4');
    if ent_1 && ent_2 && ent_3 && ent_4
        set(handles.entrenar,'Enable','on');
    end

else
    assignin('base','Elman',0);
    set(handles.entrenar,'Enable','off');

end



% --- Executes on slider movement.
function slider3_Callback(hObject, eventdata, handles)



% --- Executes during object creation, after setting all properties.
function slider3_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



% --- Executes during object creation, after setting all properties.
function imagen_CreateFcn(hObject, eventdata, handles)
imshow('Imagenes\Drum_kit.jpg')

% --- Executes on button press in pushbutton11.
function pushbutton11_Callback(hObject, eventdata, handles)

% --- Executes on slider movement.
function dinamica_Callback(hObject, eventdata, handles)
minimo_velocity=round(get(hObject,'Value'));
assignin('base','minimo_velocity',minimo_velocity);
ponderacion_dinamica=10+(1990/127)*minimo_velocity;
assignin('base','ponderacion_dinamica',ponderacion_dinamica);

% --- Executes during object creation, after setting all properties.
function dinamica_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

```
% --- Executes on slider movement.
function umbral_deteccion_Callback(hObject, eventdata, handles)
grabacion=evalin('base','grabacion')';
valor=get(hObject,'Value');
linea=valor*100*ones(1,length(grabacion));
hold off;
plot(linea,'g'); hold on;
set(gca,'ylim',[0,140]);
plot(normaliza(abs(grabacion))*100);
grid;
hold off;
assignin('base','umbral_deteccion',valor);



% --- Executes during object creation, after setting all properties.
function umbral_deteccion_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in recalcular.
function recalcular_Callback(hObject, eventdata, handles)
assignin('base','recalcular',1);

STOP_Callback(hObject, eventdata, handles)



function bpm_Callback(hObject, eventdata, handles)
bpm=get(handles.bpm,'String');
bpm=num2str(bpm);
assignin('base','bpm',bpm);



% --- Executes during object creation, after setting all properties.
function bpm_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function energia_minima_Callback(hObject, eventdata, handles)
minimo_energia=round(get(hObject,'Value'));
assignin('base','minimo_velocity',minimo_velocity);

% --- Executes during object creation, after setting all properties.
function energia_minima_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```

```
function
velocity=get_velocity(energia_media,energia_actual,energia_minima,velo
cidad_minima)
%-------------------------------------------------------------------
-----
%GET_VELOCITY    Obtiene el valor del mensaje midi Velocity.
%   Y=get_vel(ENERGIA,ENERGIA_ACTUAL,ENERGIA_MINIMA,VELOCIDAD_MINIMA)
%   obtiene el valor de la velocity Y correspondiente a funcion de
%   transferencia creada en base a la ENERGIA_MINIMA que correspondera
a la
%   VELOCIDAD_MINIMA, y la ENERGIA_MEDIA que correspondera al valor
90,
%   para el valor de ENERGIA_ACTUAL.
%
%   NOTA: No es necesario usarla con energia unicamente, puede usarse
la
%   derivada de la envolvente de la señal para determinar la
sonoridad.
% ------------------------------------------------------------------
-----
% energia_actual=log10(energia_actual);
% energia_media=log10(enegria_media);
% energia_minima=log10(energia_minima);
m=(60-velocidad_minima)/(energia_media-energia_minima);
velocity=60+m*(energia_actual-energia_media);
velocity=round(velocity);

if velocity>127
    velocity=127;
else
    if velocity<velocidad_minima
        velocity=velocidad_minima;
    end
end
function frame=mDFT(X)
%-------------------------------------------------------------------
%MDFT calcula la respuesta en magnitud de la DFT de cada segmento de
la matriz X.
%
%Entrada ----------------------------------------------------------
%   o X:      matriz Nseg x n en la que cada fila es un segmento
preprocesado.
%
%Salida -----------------------------------------------------------
%   o frame:  matriz Nseg x n con la DFT de cada segmento de X.
%
%Comentarios ------------------------------------------------------
% Como la función fft calcula la DFT en cada columna de la matriz, y X
contiene
%cada segmento en una fila, se debe transponer X para el correcto
cálculo
%de la DFT de X.
%-------------------------------------------------------------------

n=size(X,2);
c=log2(n);
if floor(c)==c% Se comprueba si los segmentos de la matriz son
potencia de 2.
    frame=abs(fft(X'));%devuelve la DFT de cada columna de la matriz.
```

```
    frame=frame';
else
    %     disp('Se utiliza zeropadding para conseguir segmentos
potencia de 2')
    n=2^(ceil(c));
    frame=abs(fft(X',n));
    frame=frame';
end

function fb=filtradoMel(M,n,fs)
%--------------------------------------------------------------------------
-----
%FILTRADOMEL Calculula una matriz con el banco de filtros mel.
%
%Entradas
%   o M:      numero de filtros.
%   o n:      longitud del frame(tamaño de la DFT).
%   o fs:     frecuencia de muestreo en Hz.
%   o fl:     frecuencia mínima(eje Mel), como fraccion de fs
%   o fh:     frecuencia maxima(eje Mel), como fraccion de fs.
% Salidas
%   o fb:     filtros
%--------------------------------------------------------------------------
-----



f0=700/fs;
fn2=floor(n/2);%ya que la FFT es par

lr=log((1+(0.5/f0)))/(M+1);
bl=n*f0*(exp([0 1 M M+1]*lr)-1);

b1 = floor(bl(1)) + 1;
b2 = ceil(bl(2));
b3 = floor(bl(3));
b4 = min(fn2, ceil(bl(4))) - 1;

pf = log(1 + (b1:b4)/n/f0) / lr;
fp = floor(pf);
pm = pf - fp;

r = [fp(b2:b4) 1+fp(1:b3)];
c = [b2:b4 1:b3] + 1;
v = 2 * [1-pm(b2:b4) pm(1:b3)];

fb = sparse(r, c, v, M, 1+fn2);
```

```
function r = MFCC(X,nF,fs)
%-------------------------------------------------------------------------
%MFCC    Función que calcula los MFCC del array de datos preprocesado
X.
% Entradas
%   o  X:      matriz Nseg x n, en la que cada fila es un segmento
preprocesado
%              de n muestras.
%   o  nF:     número de filtros de escala mel.
%   o  fs:     frecuencia de muestreo de la señal en Hz.
% Salida
%   o  coeflpcc:array Nseg x p en la que cada fila es un segmento con
p
%               coeficientes de predicción lineal
%-------------------------------------------------------------------------

n=size(X,2);%longitud del frame(tamaño de la DFT).
n2=1+floor(n/2);%ya que la FFT es par
frame=mDFT(X);%Se calcula la repuesta en magnitud de la DFT.
fb=filtradoMel(nF,n,fs);%Banco de filtros en escala mel
z=frame(:,1:n2)*fb';%se aplica el banco de filtros en escala mel a la
respuesta
%en magnitud de la FFT
r=dct(log(z));%Se aplica la transformada DCT a la salida de cada
filtro mel,
function [letra,octava]=num2letra(numero)
%-------------------------------------------------------------------------
%NUM2LETRA    obtención de la letra y octava correspondiente a numero
MIDI
%   [LETRA,OCTAVA]=num2letra(NUMERO), almacena en LETRA la letra en
%   notacion americana y en OCTAVA la octava correspondiente a ésta
%   correspondiente a la nota midi NUMERO.
%-------------------------------------------------------------------------

fila=[0:11];
for i=0:10
    matriz_midi(i+1,:)=fila+12*i;
end
letras={'C','C#','D','D#','E','F','F#','G','G#','A','A#','B'};
octavas=[-1:9];
[o,l]=find(matriz_midi==numero);
letra=letras{l};
octava=num2str(octavas(o));
function [input,env]=get_features(x,feat_elegidas)
% -------------------------------------------------------------------------
%GET_FEATURES    Caracteristicas de un sonido percusivo.
%   [INPUT,ENV]=get_features(X,FEAT_ELEGIDAS), almacena en cada
columna de
%   INPUT, cada caracteristica marcada por los indices contenidos en
%   FEAT_ELEGIDAS de la señal X.
%   ENV es la envolvente de X.
% -------------------------------------------------------------------------

%ENVOLVENTE
N=12; %numero de puntos para hacer la envolvente
env=envelope_reducida(x,N);

%ESPECTRO:
N=512; %numero de puntos para hacer la fft
spec=abs(fft(x,N)); %cambiar por spectrum si este da problemas
spec=(spec(1:end/2));
```

```
x_axis=log10([1:N/2+1]);
i=1;
if ~isempty(find(feat_elegidas==1,1))
    input_aux(i,1)=skewness(spec);
    i=i+1;
end

if ~isempty(find(feat_elegidas==2,1))
    %         input_aux(2,1)=kurtosis(spec);
    input_aux(i,1)=irregularidad(normaliza(spec));
    i=i+1;
end



if ~isempty(find(feat_elegidas==3,1))
    [~,pos]=max(spec); %el primero es el valor y el segundo la
posicion
    input_aux(i,1)=log10(pos);
    i=i+1;
end
if ~isempty(find(feat_elegidas==4,1))
    Z=100;
    if length(x)>Z
        input_aux(i,1)=mean(zcr_tramo(x(1:Z)));%mean(zcr(x,10,5));
    else
        input_aux(i,1)=mean(zcr_tramo(x));
    end
    i=i+1;
end

if ~isempty(find(feat_elegidas==5,1))
    input_aux(i,1)=centroide(spec);
    i=i+1;
end
if ~isempty(find(feat_elegidas==6,1))
    input_aux(i,1)=kurtosis(spec);
    i=i+1;
end

if ~isempty(find(feat_elegidas==7,1))
    input_aux(i,1)=centroide(env);
    i=i+1;
end

%coeficientes de Mel
if ~isempty(find(feat_elegidas>7,1))
    mfccs=MFCC(x,10,8000);
    if ~isempty(find(feat_elegidas==8,1))
        input_aux(i,1)=mfccs(1);
        i=i+1;
    end
    if ~isempty(find(feat_elegidas==9,1))
        input_aux(i,1)=mfccs(2);
        i=i+1;
    end
    if ~isempty(find(feat_elegidas==10,1))
        input_aux(i,1)=mfccs(3);
        i=i+1;
    end
    if ~isempty(find(feat_elegidas==11,1))
```

```
        input_aux(i,1)=mfccs(4);
        i=i+1;
    end
    if ~isempty(find(feat_elegidas==12,1))
        input_aux(i,1)=mfccs(5);
        i=i+1;
    end
    if ~isempty(find(feat_elegidas==13,1))
        input_aux(i,1)=mfccs(6);
        i=i+1;
    end
    if ~isempty(find(feat_elegidas==14,1))
        input_aux(i,1)=mfccs(7);
        i=i+1;
    end
    if ~isempty(find(feat_elegidas==15,1))
        input_aux(i,1)=mfccs(8);
        i=i+1;
    end
    if ~isempty(find(feat_elegidas==16,1))
        input_aux(i,1)=mfccs(9);
        i=i+1;
    end
    if ~isempty(find(feat_elegidas==17,1))
        input_aux(i,1)=mfccs(10);
        i=i+1;
    end
end
input=input_aux;
function c=centroide(s)
%-------------------------------------------------------------------
%CENTROIDE    posicion del centroide
%   C=centroide(S), almacena en C la posicion en el eje de abcisas del
%   centroide de la funcion S.
% ------------------------------------------------------------------
area=sum(s);

c=sum(s.*[1:length(s)])/area;
function y=irregularidad(x)
% ------------------------------------------------------------------
%IRREGULARIDAD    Irregularidad de una funcion
%    Y=irregularidad(X), Almacena en la variable Y el valor de la
%    irregularidad de la funcion X, como cantidad de variaciones
definidas
%    por su derivada.
% ------------------------------------------------------------------

deriv=abs(diff(x));
y=sum(deriv);
function z=zcr_tramo(x)
% ------------------------------------------------------------------
% zcr_tramo    tasa de cruces por cero
%       Z=zcr_tramo(X) obtiene la tasa de cruces por cero de la señal
X,
%       como el numero de cruces por cero entre la longitud del tramo.
% ------------------------------------------------------------------
z=0;
X=length(x);
for i=1:X-1
    if (x(i)< 0 && x(i+1)>0) || (x(i)> 0 && x(i+1)<0)
        z=z+1;
```

```
    end
end
z=z/X;
function y=eval_features(inputs)
% ---------------------------------------------------------------------
%EVAL_FEATURES    Evaluacion interclase de las caracteristicas.
%   Y=eval_features(INPUTS)devuelve un indicador de las filas
%   correspondientes a las caracteristicas que mas discriminacion
%   interclase encuentran en la matriz de entrada INPUTS.
%
%   INPUTS: cell array de una fila que almacena en cada columna una
matriz
%   correspondiente a las caracteristicas extraidas para cada clase.
%   Y=array que indica los indices de las caracteristicas elegidas.
% ---------------------------------------------------------------------

% N=5; %numero de features que se van a escoger por maximo
y=[];
if ~isempty(inputs{1,1})
    inp1=cell2mat(inputs{1,1});
end
if ~isempty(inputs{1,2})
    inp2=cell2mat(inputs{1,2});
end
if ~isempty(inputs{1,3})
    inp3=cell2mat(inputs{1,3});
end
if ~isempty(inputs{1,4})
    inp4=cell2mat(inputs{1,4});
end

%evaluacion por el algoritmo Relieff
umbral_seleccion=0.16;

grupo(1:40,1)='1';
grupo(41:80,1)='2';
grupo(81:120,1)='3';
grupo(121:160,1)='4';

all_inputs=[inp1,inp2,inp3,inp4];
[ranked,peso]=relieff(all_inputs',grupo,10);

i=1;
while peso(ranked(i))>umbral_seleccion
    y(i)=ranked(i);
    i=i+1;
end
j=1;
while isempty(y) && j<=6
    umbral_seleccion=umbral_seleccion*0.8;
    i=1;
    while peso(ranked(i))>umbral_seleccion
        y(i)=ranked(i);
        i=i+1;
    end
    j=j+1;
end
if j==6
    y=ranked(1:3);
end
```

```
function y=normaliza_mat(matriz)
%-------------------------------------------------------------------
%NORMALIZA_MAT    Normalizacion de una matriz.
%   Y=normaliza_mat(MATRIZ) normaliza las filas de la matriz MATRIZ,
%   almacenandolas en Y.
%-------------------------------------------------------------------
[M,N]=size(matriz);
y=zeros(M,N);
for i=1:N
    y(:,i)=normaliza(matriz(:,i));
end

function y=envelope_reducida(x,N)
%-------------------------------------------------------------------
%ENVELOPE_REDUCIDA     Obtención de la envolvente de una señal de
audio.
%   Y=envelope_reducida(X,N), obtiene la envolvente Y de la señal de
%   entrada X, tomando ventanas de N puntos con solapamiento del 50%.
Cada
%   valor de Y representa N/2 muestras de X.
% -------------------------------------------------------------------
M=length(x);
y=zeros(1,round(M/N));
j=1;
for i=N/2+1:N/2:M-N/2
    y(j)=max(abs(x(i-N/2:i+N/2)));
    j=j+1;
end

function nota=instrumento2nota(instrumento)
% -------------------------------------------------------------------
-----
%INSTRUMENTO2NOTA    Traduccion de numero de nota midi a notacion
americana
%   NOTA=instrumento2nota(INSTRUMENTO) obtiene la notacion americana
en la
%   variable de salida NOTA, que define el numero de nota midi
%   correspondiente a ISTRUMENTO.
% -------------------------------------------------------------------
-----
n=[0 12 24 36 48 60 72 84 96 108 120];
if max(instrumento==0+n)
    [~,octava]=max(instrumento==0+n);
    octava=octava-2;
    nota=['C',num2str(octava)];
    %     set(handles.nota1,'String',nota);
else if max(instrumento==1+n)
        [~,octava]=max(instrumento==1+n);
        octava=octava-2;
        nota=['C#',num2str(octava)];
    else if max(instrumento==2+n)
            [~,octava]=max(instrumento==2+n);
            octava=octava-2;
            nota=['D',num2str(octava)];
        else if max(instrumento==3+n)
                [~,octava]=max(instrumento==3+n);
                octava=octava-2;
                nota=['D#',num2str(octava)];
            else if max(instrumento==4+n)
                    [~,octava]=max(instrumento==4+n);
                    octava=octava-2;
```

```
                    nota=['E',num2str(octava)];
                else if max(instrumento==5+n)
                        [~,octava]=max(instrumento==5+n);
                        octava=octava-2;
                        nota=['F',num2str(octava)];
                    else if max(instrumento==6+n)
                            [~,octava]=max(instrumento==6+n);
                            octava=octava-2;
                            nota=['F#',num2str(octava)];
                        else if max(instrumento==7+n)
                                [~,octava]=max(instrumento==7+n);
                                octava=octava-2;
                                nota=['G',num2str(octava)];
                            else if max(instrumento==8+n)
                                    [~,octava]=max(instrumento==8+n);
                                    octava=octava-2;
                                    nota=['G#',num2str(octava)];
                                else if max(instrumento==9+n)

[~,octava]=max(instrumento==9+n);
                                        octava=octava-2;
                                        nota=['A',num2str(octava)];
                                    else if max(instrumento==10+n)
                                            [~,octava]=...

max(instrumento==10+n);
                                            octava=octava-2;

nota=['A#',num2str(octava)];
                                        else if max(instrumento==11+n)
                                                [~,octava]=...

max(instrumento==11+n);
                                                octava=octava-2;

nota=['B',num2str(octava)];
                                            end
                                        end
                                    end
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
```

```
function escribe_fichero(file,mensaje,segundo_evento,
nota,octava,velocity)
% ------------------------------------------------------------------
%ESCRIBE_FICHERO  Escritura de evento en fichero de midi texto
%   escribe_fichero(FILE,MENSAJE,SEGUNDO_EVENTO,NOTA,OCTAVA,VELOCITY)
escribe en
%   el fichero con handle FILE, el evento definido en MENSAJE, que se
%   produce en el sengundo SEGUNDO_EVENTO, con nota midi NOTA, octava
%   OCTAVA y velocidad (volumen) VELOCITY.
%
%   MENSAJE: o  "On Note"
%            o  "Off Note"
%            o  "Pitch Wheel"
%            o  "Aftertouch"
%   NOTA: nota midi en notacion americana. Ejemplo A#3 LA sostenido de
la
%   tercera octava
%   OCTAVA: Numero entero entre -2 y 8.
%   VELOCITY: Valor de el volumen del evento entre 0 y 127;
% ------------------------------------------------------------------

natural=fix(segundo_evento);
bloque=fix(natural/4);
segundos=natural-bloque*4;%4 compases se consideran un bloque
decimales=segundo_evento-fix(segundo_evento);
decimales_todos=(decimales);
decimales=(num2str(decimales_todos));
L=length(decimales);
if length(decimales)>5
    decimales=decimales(1:5);
end
L=length(decimales);
switch L

    case 3
        mil=[decimales(3),'0','0'];
    case 4
        mil=[decimales(3),decimales(4),'0'];
    case 5
        if decimales_todos>=5 && decimales_todos(6)>5
            decimales(5)=decimales(5)+1;
        end
        mil=[decimales(3),decimales(4),decimales(5)];
    otherwise
        mil=['0','0','0'];

end
bloque=bloque+1;
segundos=segundos+1;

switch mensaje(1:2)
    case 'On'
        cadena = 'On Note   ';
    case 'Of'
        cadena = 'Off Note  ';
    case 'Af'
        cadena = 'Aftertouch';
    case 'Pi'
        cadena = 'Pitch Wheel';
```

```
end

fprintf(file,' %3d:%3d:%3s',bloque,segundos,mil);
fprintf(file,' |%8s |',cadena);
fprintf(file,' chan=1 | ');
if strcmp(mensaje(1:2),'Pi')
    fprintf(file,' bend= %3d \n',velocity);
else
    fprintf(file,' pitch= %2s %2s | ',nota,octava);
    if strcmp(mensaje(1:2),'Af')
        fprintf(file,'press= %3d \n',velocity);
    else
        fprintf(file,'vol=   %3d \n',velocity);
    end
end

function [estado,energia]=estado_tomar_inputs(x,estado_anterior,
umb_derivada,umbral_energia,energ_max)
%-------------------------------------------------------------------------
%ESTADO_TOMAR_INPUTS   Decide el estado de la ventana 'x'
%
[ESTADO,ENERGIA]=estado_tomar_inputs(X,ESTADO_ANTERIO,UMB_DERIVADA,...
%   UMBRAL_ENERGIA,ENERG_MAX).
%
%   VARIABLES DE ENTRADA:
%       o X: tramo de señal a evaluar.
%       o ESTADO_ANTERIOR: estado correspondiente a la anterior
ventana.
%       o UMB_DERIVADA: umbral de la derivada
%       o UMBRAL_ENERGIA: umbral de energia.
%       o ENERG_MAX: Energia maxima almacenada a lo largo de las
anteriores
%       ventanas catalogadas como 'inicio'
%
%   VARIABLES DE SALIDA
%       o ESTADO: representa el estado en leque se encuentra la
ventana,
%       puede ser:
%               -'ruido': la ventana corresponde a ruido.
%               -'inicio': la ventana corresponde a un inicio de
golpe.
%               -'fin': la ventana corresponde a un fin de golpe.
%       o ENERGIA: energia maxima dada en la ventana.
%-------------------------------------------------------------------------


L=length(x);
derivada=max(diff(x));
energia=max(sum(x.^2)/L);

if strcmp(estado_anterior,'ruido')==1 ||
strcmp(estado_anterior,'fin')==1
    if energia>umbral_energia && derivada>umb_derivada
        estado='inicio';
    else
        estado=estado_anterior;
    end
else %si venimos de un 'inicio'
    if energia<0.2*energ_max
        estado='fin';
```

```
    else
        estado=estado_anterior;
    end
end
```